Санкт–Петербургский государственный университет

ГУЛЯЕВ Никита Алексеевич

Выпускная квалификационная работа

Один класс дифференциальных игр преследования

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2017 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Системный анализ, исследование операций и управление»

Научный руководитель:
профессор, кафедра математической теории игр
и статистических решений, д.ф. - м.н. Петросян Леон Аганесович

Рецензент:
доцент, кафедра моделирования экономических
систем, к.ф. - м.н. Ковшов Александр Михайлович

Санкт-Петербург

2021 г.

Содержание

Введение

Теория игр преследования изучает ситуации противостояния двух классов игроков - преследователей и убегающих. Преследователи стремятся как можно быстрее поймать всех убегающих (здесь поимка - суть сближение на заданное расстояние, например, равное длине вытянутой руки). Убегающие, напротив, стремятся как можно дольше избегать поимки, если возможно - не допускать ее в принципе. Основная задача теории игр преследования - вывести оптимальные стратегии (суть уравнения, выражающие зависимость скорости рассматриваемого игрока от местоположения каждого из игроков и, возможно, скорости оппонента) для каждого игрока. С другой стороны, многочисленные варианты игр преследования могут переопределять понятие стратегии игрока произвольным образом.

У игр преследования выделено значительное число классов и вариаций. Делят их, в том числе, по количеству преследователей и убегающих (наиболее простой и изученный вариант - игры с одним убегающим и одним преследователем); по размерности пространства, в котором проходит погоня (будь то преследование на плоскости, в трехмерном пространстве или даже в пространствах высшей размерности, в которых игра преследования носит уже более абстрактный, оторванный от реальности смысл); по ограниченности этого пространства (иначе говоря, наличие, расположение и форма “стен”, препятствующих движению).

Выделяется крупная категория игр преследования с «линией жизни» - в них цели игроков модифицируются: убегающий, в целом, стремится пересечь линию, называемую «линией жизни», гарантирующую ему неприкосновенность, а преследователь - поймать убегающего до того, как это случится. Пример, иллюстрирующие смысл «линии жизни» - берег в ситуации спасения от агрессивно настроенных морских обитателей.

Наконец, исследуются даже игры преследования в вихревых пространствах, где “ветер” словно помогает игрокам двигаться в определенных направ-

лениях.

Приложения теории дифференциальных игр преследования довольно широки. Игры преследования на плоскости могут использоваться для корректировки программ дрессировки охотничьих собак. Если на плоскость добавить препятствия, становится возможно эффективное моделирование процессов поимки сбегающих заключенных, автомобильной погони.

К играм преследования в пространстве имеется большой интерес у военной отрасли, они позволяют разрабатывать высокоточные программы управления самонаводящимися ракетами. Становится возможным точный расчет оптимальных траекторий и для более сложных физических условий - переход из воздушной среды в водную, движение воздушных масс, уклонение от препятствий и огибание специальных зон. Сюда же относятся и игры преследования с «линией жизни», ведь они позволяют учитывать расстояние, на котором сигналы достигают управляемых ракет.

Различные вариации игр преследования, с другой стороны, в последнее время также используются и для разработки продвинутого ИИ в видеоиграх различных жанров, позволяя создавать ботов, моделирующих поведение реальных игроков с крайне высокой степенью точности.

Постановка задачи

В данной работе рассматривается конкретный класс игр преследования на плоскости - игры преследования в круге. Его характерной чертой является особое правило, запрещающее любому из игроков покидать пределы заранее заданного круга.

Работа мотивирована следующей глобальной задачей. Положим, что в игре участвуют два игрока - преследователь и убегающий, местоположения которых в каждый момент времени определяются двумерными векторами, задающих координаты точек на плоскости. Пусть заданы две точки на плоскости, принадлежащие конкретному кругу и определяющие начальные место-

положения преследователя и убегающего. Пусть также известны максимальные значения, которые могут принимать модули скоростей игроков. Будем считать допустимыми движения игроков, удовлетворяющие следующим двум условиям:

- Модуль скорости в любой момент времени не превышает заданного для данного игрока максимального значения;
- Местоположение игрока в любой момент времени находится в заданном круге, общем для обоих игроков.

Поскольку движение игрока будет зависеть от движения его противника, будем считать, что эти требования должны быть справедливы для всех допустимых движений соперника.

Требуется определить, какой вид должно принять допустимое движение преследователя, определяемое зависимостью двумерного вектора скорости преследователя от текущего местоположения преследователя, текущего местоположения убегающего и текущей скорости убегающего, обеспечивающее минимальное время поимки (под которой понимается совпадение местоположений убегающего и преследователя в некоторый момент времени) при произвольном допустимом движении убегающего. Аналогично, необходимо определить допустимое движение убегающего, выражаемое как зависимость местоположения убегающего от текущих местоположений игроков, доставляющее максимум времени поимки при произвольном допустимом движении преследователя.

В рамках данной работы интерес представляет не само решение дифференциальной игры, поскольку данная задача обладает высокой сложностью, а определение свойств, которыми оно должно обладать. С другой стороны, ввиду практической направленности работы, основной задачей является разработка системы, позволяющей решать описанную выше задачу выявления качеств достаточно хороших стратегий.

Итого, задача данного исследования ставится следующим образом:

- Определение дискретного аналога игры преследования в круге;
- Разработка системы, реализующий данный аналог, и позволяющей производить симуляции для различных комбинаций стратегий (законов движения) игроков;
- Испытание данной системы для реальных задач сравнения эффективности стратегий;
- Разработка алгоритмов перебора стратегий убегающего для выявления наиболее близких к оптимальной (или оптимальным, если таковых несколько);
- Проведение экспериментов при помощи разработанных алгоритмов перебора.

Обзор литературы

В ходе исследования было проведено ознакомление с рядом источников, принадлежащих данной области.

В первую очередь это были [12], [11] и [10]. В [12] приводится общий обзор основных определений теории дифференциальных игр и перечисление ряда нерешенных проблем. В [11], как и в [10] приводится более обстоятельное описание всех основных направлений теории игр преследования, при этом в [11] используется более геометрический подход, в то время как в [10] - более аналитический. В рамках данной работы [11] используется больше как источник основополагающих определений, в то время как [10] - как учебник, содержащий формулировки и доказательства важных теорем.

Впервые понятие дифференциальной игры определено в [7], а самое широкое определение, аппелирующее к положениям теории игр, приведено в [8].

В проведенном исследовании оно позволило корректно определить дискретный аналог игры преследования, вокруг которого проводилась дальнейшая работа. Помимо этого, важные положения теории дифференциальных игр вводятся в [9].

Ряд современных источников не нашел применения в данной работе, однако послужил общему представлению о состоянии теории дифференциальных игр и может быть использован в дальнейших исследованиях. В их число входит [6], в котором представлены определения и описания большого числа классов игр преследования, а также приведен процесс получения решения в некоторых из них. Широкий обзор современного состояния теории дифференциальных игр приведен в [5]. Там же рассмотрены все основные подходы к решению различных классов игр преследования. Отдельный класс, в котором скорости двух игроков равны, а также на плоскости имеется круглое препятствие, разобран в [3], в котором приводятся условия для победы и выигрышные стратегии для каждого из игроков. Класс игр с несколькими преследователями и несколькими убегающими рассматривается в [2].

Для проведения работы также использовались вспомогательные источники. В [4] имеется важная глава с разбором треугольного распределения, которое в рамках данной работы было использовано для разработки стратегии уклонения с шумом. В [1] приводится описание критерия Пейджа непараметрического двухфакторного анализа с упорядоченными альтернативами, использованного для проверки статистических гипотез о сравнительной эффективности стратегий.

Глава 1. Игра простого преследования на плоскости

1.1. Математическая модель игры простого преследования

Определим дифференциальную игру Г преследования на плоскости с одним преследователем и одним убегающим аналогично определению, данному в [7] и в соответствии с обобщенным определением, данным в [8].

Рассмотрим плоскость, на которой ведется игра Г. Введем на ней ДПСК с осями O_x и O_y и центром O. Пусть местоположения преследователя и убегающего соответственно в момент времени t описываются следующими вектор-функциями:

$$\begin{cases} P(t) = (x_P(t), y_P(t)), \\ E(t) = (x_E(t), y_E(t)). \end{cases} \tag{1}$$

Зафиксируем $P(0), E(0)$ - местоположения игроков в начальный момент времени.

Пусть также

$$\begin{cases} u(t) = (u_x(t), u_y(t)), \\ v(t) = (v_x(t), v_y(t)) \end{cases} \tag{2}$$

- вектор-функции скоростей преследователя и убегающего, для которых верны неравества:

$$\begin{cases} \forall t \ |u(t)| \leq \alpha, \\ \forall t \ |v(t)| \leq \beta. \end{cases} \tag{3}$$

При этом также положим, что выполняется следующее соотношение:

$$\alpha > \beta > 0 \tag{4}$$

Данное неравенство означает, что преследователь должен быть строго быстрее убегающего.

Будем полагать, что движение игроков происходит согласно следующим векторным дифференциальным уравнениям:

$$\begin{cases} \dot{P}(t) = u(t), \\ \dot{E}(t) = v(t). \end{cases} \tag{5}$$

Или, разрешая систему ОДУ (5) с учетом начальных условий:

$$\begin{cases} P(t) = P(0) + \int_0^t u(s)\,ds, \\ E(t) = E(0) + \int_0^t v(s)\,ds. \end{cases} \tag{6}$$

В каждый момент времени игрок-убегающий (далее - У) выбирает величину и направление вектора скорости v, опираясь на сведения о местоположениях двух игроков в текущий момент времени: P и E соответственно. В соответствии с выбранной убегающим скоростью v, а также местоположениями P и E игроков в пространстве, игрок-преследователь (далее - П), в свою очередь, выбирает величину и направление вектора скорости u.

Формально множества стратегий игроков можно определить, с учетом (3), следующим образом:

$$\begin{cases} U = \{u(P,E,v) \mid P,E \in C \wedge |v| \leq \beta \implies |u(P,E,v)| \leq \alpha\}, \\ V = \{v(P,E) \mid P,E \in C \implies |v(P,E)| \leq \beta\}. \end{cases} \tag{7}$$

При фиксированной функции выигрыша $H(u,v)$, описанные выше усло-

вия (1)-(4), (6) вкупе с определениями множеств (7) задают антагонистическую игру преследования $\Gamma =< V, U, H >$ с параметрами α, β, $E(0)$, $P(0)$.

1.2. Некоторые способы задания функции выигрыша

Перечислим основные способы задать функцию выигрыша $H(u, v)$.

Для упрощения записи будем использовать функции местоположений игроков $P(t), E(t)$, для каждой конкретной игры преследования выводымые из функций u, v согласно (6) - системе неявных интегральных уравнений от функций $P(t), E(t)$.

1. Игра на быстродействие

$$H(u, v) = \min_{P(T)=E(T)} T \tag{8}$$

Здесь, если при использовании выбранных игроками стратегий игроку П не удается догнать игрока У, то выигрыш игрока У принимается равным ∞, соответственно, выигрыш игрока П - равным $-\infty$.

Данный вид функции выигрыша соответствует ситуации, в которой целью преследователя является поимка убегающего за как можно меньшее время, а целью убегающего, напротив, уклонение от погони на протяжении как можно большего времени.

2. Максимизация расстояния в момент T

$$H(u, v) = |E(T) - P(T)| \tag{9}$$

Посредством определения функции выигрыша данным способом, описывается случай, когда убегающему требуется спустя заданное время T оказаться как можно дальше от преследователя. Здесь цель преследователя - спустя заданное время T оказаться как можно ближе к убегающему.

3. Максимизация минимального расстояния за время T

$$H(u,v) = \min_{0 \le t \le T} |E(t) - P(t)| \tag{10}$$

В играх с данной функцией выигрыша цель убегающего - гарантировать себе как можно большее расстояние, ближе чем на которое за время T преследователь не приблизится. Цель преследователя - как можно сильнее сократить расстояние с убегающим за время, меньшее или равное T.

1.3. Некоторые стратегии игроков

Для начала введем подвижную систему координат O_{ab} с центром в точке O и осями O_a и O_b такими, что O_a в каждый момент времени сонаправлена с вектором $E(t) - P(t)$, а O_b перпендикулярна O_a и направлена таким образом, что пара ортов $\vec{a}$ и $\vec{b}$ положительно ориентирована (здесь $\vec{a} \uparrow\uparrow O_a$, $\vec{b} \uparrow\uparrow O_b$).

Теперь приведем стратегии преследования по погонной линии и параллельного сближения, описанные в [10]:

1. Преследование по погонной линии

$$u(E,P,v) = \alpha * \frac{E-P}{|E-P|} = \alpha\vec{a} \tag{11}$$

2. Параллельное сближение

$$\begin{cases} u_a(E,P,v) = \sqrt{\alpha^2 - v_b^2(t)}, \\ u_b(E,P,v) = v_b(t). \end{cases} \tag{12}$$

Здесь u_a, u_b - проекции вектора u, а v_a, v_b - проекции вектора v на оси O_a и O_b.

В [10] показано, что в играх простого преследования на быстродействие стратегия параллельного сближения является оптимальной, а при дополнении определения игры условием $u = u(P,E)$ (т. е. независимостью u от v), оптимальной стратегией преследования становится преследование по погон-

ной линии.

С другой стороны, в [10] также приводится доказательство того, что в игре на быстродействие оптимальной стратегией убегающего является движение по прямой в направлении вектора $E(t) - P(t)$ с максимальной скоростью, описываемое уравнением:

$$v(E, P) = \beta * \frac{E - P}{|E - P|} = \beta \vec{a} \tag{13}$$

1.4. Дискретная игра простого преследования

Будем также рассматривать дискретный аналог игры простого преследования. По сравнению с обычной игрой простого преследования (1)-(4), (6), формула (6) отклоняется. Изменение местоположений игроков E и P проходит по следующему алгоритму:

0. В начале игры принимаем $t = 0$;

1. Игрок У выбирает вектор перемещения $v(t)$ в соответствии с местоположениями игроков $P(t)$ и $E(t)$ таким образом, чтобы $|v(t)| = \beta$;

2. Игрок П выбирает вектор перемещения $u(t)$ в соответствии с выбранным игроком У вектором $v(t)$ и местоположениями игроков $P(t)$ и $E(t)$ таким образом, чтобы $|u(t)| \leq \alpha$;

3. Новые местоположения игроков вычисляются по правилу $P(t+1) = P(t) + u(t)$ и $E(t+1) = E(t) + v(t)$;

4. Переход к первому шагу с t = t + 1.

При этом вектор-функции $P(t)$ и $E(t)$ определяются только в точках $t \in \mathbb{N} \cup \{0\}$. Аналогично $v(t)$ и $u(t)$ задаются на том же множестве t. Ввиду этого, множества управлений $v(t)$ и $u(t)$ преобразуются к следующему виду:

$$\begin{cases} U = \{(u(0), u(1), ...) \mid \forall t \ |u(t)| \leq \alpha\}, \\ V = \{(v(0), v(1), ...) \mid \forall t \ |v(t)| = \beta\}. \end{cases} \tag{14}$$

В отличие от множеств управлений (14), вид множеств стратегий (7) не изменяется при переходе к дискретному случаю.

Приведенные выше способы задания функции выигрыша (8)-(10) не требуют переформулировки. Также не требует модификации и стратегия убегающего (13). С другой стороны, у описанных стратегий преследователя (11)-(12) появляется недостаток: при достаточной близости к убегающему, преследователь, двигаясь в нужном направлении с максимальной скоростью, пробегает мимо убегающего, что позволяет убегающему уклоняться от погони бесконечно. Эта проблема может быть исправлена следующей модификацией стратегий преследователя:

1. Преследование по погонной линии

$$u(t) = \min\{|E(t) + v(t) - P(t)|; \alpha\} \frac{E(t) + v(t) - P(t)}{|E(t) + v(t) - P(t)|} \tag{15}$$

2. Параллельное сближение

$$\begin{cases} u_a(t) = \sqrt{\min^2\{|E(t) + v(t) - P(t)|; \alpha\} - v_b^2(t)}, \\ u_b(t) = v_b(t). \end{cases} \tag{16}$$

Для $|E(t)+v(t)-P(t)| \geq \alpha$ описанные модификации (15)-(16) совпадают с оригинальными стратегиями преследователя (11)-(12). При этом стратегия преследования по погонной линии (15) приобретает зависимость от v(t).

1.5. Игра с «линией жизни»

Согласно [10], чтобы определить игру с «линией жизни», достаточно дополнить определение (1)-(4), (6) особым выбором функции выигрыша:

$$\begin{cases} H(u(t), v(t)) = -1, \ t_P < t_S^E \wedge t_P < \infty, \\ H(u(t), v(t)) = 0, \ t_P = t_S^E = \infty, \\ H(u(t), v(t)) = 1, \ t_P > t_S^E \wedge t_S^E < \infty. \end{cases} \tag{17}$$

Где $S \subset \mathbb{R}^2$ - некоторое фиксированное множество и:

$$\begin{cases} t_P = \min\{t : P(t) = E(t)\}, \\ t_S^E = \inf\{t : E(t) \in S\}. \end{cases} \tag{18}$$

Исходя из способа задания функции выигрыша, цель убегающего - достигнуть множества S, не будучи настигнутым преследователем а цель преследователя - догнать убегающего до достижения им множества S.

Глава 2. Игра простого преследования в круге

2.1. Математическая модель игры преследования в круге

Предположим, что в игре преследования (1)-(4), (6) с множествами стратегий игроков (7) оба игрока не могут покидать пределы некоторого круга C. Для простоты положим, что центр этого круга находится в центре координат O (до этого центр координат определялся произвольно). Обозначим радиус круга C как R. Тогда определение описанной игры преследования Γ_C примет следующий вид:

$$\begin{cases} P(t) = (x_P(t), y_P(t)), \\ E(t) = (x_E(t), y_E(t)), \\ \dot{P}(t) = u(t) = (u_x(t), u_y(t)), \\ \dot{E}(t) = v(t) = (v_x(t), v_y(t)), \\ P(0) = P_0,\ |P_0| \leq R, \\ E(0) = E_0,\ |E_0| \leq R, \\ X = \{\dot{P}(t) \mid \forall t\ |\dot{P}(t)| \leq \alpha\ \wedge\ |P(t)| \leq R\}, \\ Y = \{\dot{E}(t) \mid \forall t\ |\dot{E}(t)| \leq \beta\ \wedge\ |E(t)| \leq R\}, \\ \alpha > \beta > 0, \\ \Gamma_C =< Y, X, H > . \end{cases} \tag{19}$$

Здесь H - функция выигрыша, определяющаяся согласно одному из уравнений (8)-(10) с использованием перехода (6), α, β, R, P_0, E_0 - параметры семейства игр Γ_C

Аналогично определяется дискретный аналог игры простого преследования в круге Γ'_C с множествами стратегий игроков (7):

$$\begin{cases} P(t) = (x_P(t), y_P(t)), \\ E(t) = (x_E(t), y_E(t)), \\ u(t) = (u_x(t), u_y(t)), \\ v(t) = (v_x(t), v_y(t)), \\ P(t+1) = P(t) + u(t), \\ E(t+1) = E(t) + v(t), \\ P(0) = P_0,\ |P_0| \leq R, \\ E(0) = E_0,\ |E_0| \leq R, \\ X = \{(u(0), u(1), ...) \mid \forall t\ |u(t)| \leq \alpha\ \wedge\ |P(t)| \leq R\}, \\ Y = \{(v(0), v(1), ...) \mid \forall t\ |v(t)| = \beta\ \wedge\ |E(t)| \leq R\}, \\ \alpha > \beta > 0, \\ \Gamma'_C =< Y, X, H > . \end{cases} \tag{20}$$

2.2. Допустимость стратегий в игре преследования в круге

2.2.1. Допустимость стратегий для игры Γ_C

Рассмотрим стратегии (11)-(13) применительно к игре преследования в круге (19). Проверить соответствующие условия на $P(t)$, $E(t)$ можно предварительно использовав формулы перехода (6):

$$\begin{cases} P(t) = P(0) + \int_0^t u(s)\,ds, \\ E(t) = E(0) + \int_0^t v(s)\,ds. \end{cases}$$

Проверим выполнение условий $u(t) \in X$ и $v(t) \in Y$ для $u(t)$, $v(t)$, полученных из формул (11)-(13), и X, Y, определяемых из (19).

Утверждение 1. *В игре Γ_C для любого движения убегающего $E(t)$, удовлетворяющего условиям $\forall t\ |E(t)| \le R$ и $\forall t\ |\dot{E}(t)| \le \beta$, движение преследователя $P(t)$, соответствующее стратегии преследования по погонной линии (11), удовлетворяет условиям $\forall t\ |P(t)| \le R$ и $\forall t\ |\dot{P}(t)| \le \alpha$.*

Доказательство.

$$\dot{P}(t) = (P(0) + \int_0^t u(s)\,ds)' = u(t)$$

$$|\dot{P}(t)| = |u(t)| = |\alpha\vec{a}(t)| = \alpha|\vec{a}(t)| = \alpha \le \alpha$$

Поскольку круг обладает свойством выпуклости:

$$\forall P(t), E(t)\ (|P(t)| \le R \wedge |E(t)| \le R) \implies \forall \gamma \in [0;1]\ P(t) + \gamma(E(t) - P(t)) \le R$$

Следовательно,

$$\forall P(t), E(t) : |P(t)| \le R\ \wedge\ |E(t)| \le R\ \ \exists \delta t :\ \forall \gamma \in [0; \delta t]\ P(t) + \gamma u(t) \le R$$

Объединяя с $|P(0)| \le R$, $|E(t)| \le R$, получаем искомое:

$$\forall t\ |P(t)| \le R$$

□

Утверждение 2. *Существуют α, β, R, а также точки $E(0)$, $P(0)$, вектор-функция скорости убегающего $v(t)$, удовлетворяющие условиям игры Γ_C, и момент времени t_1, такие, что точка $P(t_1)$, полученная применением стратегии параллельного сближения (12), удовлетворяет условию $|P(t_1)| > R$.*

Доказательство. Пусть $\alpha = \frac{\sqrt{5}}{2}$, $\beta = 1$, $R = 1$, $E(0) = (0, \frac{\sqrt{3}}{2})$, $P(0) = (-0.5, \frac{\sqrt{3}}{2})$, $t_1 = 1 - \frac{\sqrt{3}}{2}$, $v(t)$ такова, что $\forall t \in [0; t_1]$ $v(t) = (0, 1)$.

Тогда $E(0) - P(0) = (0.5, 0)$. Следовательно, $u(0) = (0.5, 1)$, а так как $v(t)$ неизменна на $[0; t_1]$, то $\forall t \in [0; t_1]$ $u(t) = (0.5, 1)$.

Но тогда $P(t_1) = (-0.5, \frac{\sqrt{3}}{2}) + (0.5, 1) * (1 - \frac{\sqrt{3}}{2}) = (\frac{-\frac{\sqrt{3}}{2}}{2}, 1) = (-\frac{\sqrt{3}}{4}, 1)$.

Отсюда: $|P(t_1)| = \sqrt{(\frac{\sqrt{3}}{4})^2 + 1} > \sqrt{1} = 1 = R$.

□

Утверждение 3. *Существуют α, β, R, а также точки $E(0)$, $P(0)$, вектор-функция скорости преследователя $u(t)$, удовлетворяющие условиям игры Γ_C, и момент времени t_1, такие, что точка $E(t_1)$, полученная применением стратегии уклонения (13), удовлетворяет условию $|E(t_1)| > R$.*

Доказательство. Пусть $\alpha = 2$, $\beta = 1$, $R = 1$, $E(0) = (1, 0)$, $P(0) = (-1, 0)$, $t_1 = 1$, $u(t)$ такова, что $\forall t \in [0; t_1]$ $u(t) = (2, 0)$.

Тогда $E(0) - P(0) = (2, 0)$. Следовательно, $v(0) = (1, 0)$, и так как $v(0)$ и $u(t)$ для $t \in [0; t_1]$ параллельны прямой, соединяющей начальные местоположения игроков, а также основываясь на (13) получаем: $\forall t \in [0; t_1]$ $v(t) = (1, 0)$.

Но тогда $E(t_1) = (1, 0) + (1, 0) * 1 = (2, 0)$.

Отсюда $|E(t_1)| = 2 > 1 = R$.

□

2.2.2. Допустимость стратегий для игры Γ'_C

Рассмотрим теперь игру Γ'_C, определяемую системой (20), с позиции допустимости стратегии игрока У (13) и стратегий игрока П (15), (16), полу-

ченных для игры на плоскости в неограниченном пространстве.

Утверждение 4. *В игре Γ'_C для любого местоположения $E(t)$ и скорости $v(t)$ убегающего, для которых $|v(t)| = \beta \wedge |E(t) + v(t)| \leq R$, движение преследователя $u(t)$, соответствующее стратегии преследования по погонной линии (15), удовлетворяет условиям $|P(t) + u(t)| \leq R$ и $|u(t)| \leq \alpha$.*

Доказательство.

$$|u(t)| = |\min\{|E(t) + v(t) - P(t)|; \alpha\}\frac{E(t) + v(t) - P(t)}{|E(t) + v(t) - P(t)|}| \leq \alpha * 1 = \alpha$$

$$|P(t) + u(t)| = |P(t) + \min\{|E(t) + v(t) - P(t)|; \alpha\}\frac{E(t) + v(t) - P(t)}{|E(t) + v(t) - P(t)|}|$$

$$|P(t) + u(t)| = \begin{cases} |P(t) + (E(t) + v(t) - P(t))|, |E(t) + v(t) - P(t)| < \alpha, \\ |P(t) + \alpha\frac{(E(t)+v(t))-P(t)}{|(E(t)+v(t))-P(t)|}|, |E(t) + v(t) - P(t)| \geq \alpha. \end{cases}$$

$$|P(t) + u(t)| = \begin{cases} |E(t) + v(t)|, |E(t) + v(t) - P(t)| < \alpha, \\ |P(t) + \gamma((E(t) + v(t)) - P(t))|, |E(t) + v(t) - P(t)| \geq \alpha, \end{cases}$$

где $\gamma \in (0; 1]$

Верхняя часть склейки меньше R по условию, а нижняя часть - в силу принадлежности C ввиду $P(t) \in C$, $E(t) + v(t) \in C$ и выпуклости C. Отсюда $|P(t) + u(t)| < R$

□

Утверждение 5. *Существуют α, β, R, а также точки $E(t)$, $P(t)$ и вектор скорости убегающего $v(t)$, удовлетворяющие условиям игры Γ'_C, такие, что точка $P(t+1)$, полученная применением стратегии параллельного сближения (16), удовлетворяет условию $|P(t + 1)| > R$.*

Доказательство. Пусть $\alpha = \frac{\sqrt{5}}{2}$, $\beta = 1$, $R = 1$, $E(t) = (0, 0)$, $P(t) = (-1, 0)$, $v(t) = (0, 1)$.

$|E(t) + v(t) - P(t)| = |(0,0) + (0,1) - (-1,0)| = |(1,1)| = \sqrt{2} = \frac{\sqrt{8}}{2} > \frac{\sqrt{5}}{2} = \alpha$

Так как $E(t) - P(t) = (1,0)$, подвижная система координат совпадает с неподвижной. Поэтому: $|P(t+1)| = |P(t) + (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t))| = |(-1,0) + (0.5,1)| = |(-0.5,1)| = \frac{\sqrt{5}}{2} > 1 = R$

□

Утверждение 6. *Существуют α, β, R, а также точки $E(t)$, $P(t)$ и вектор скорости преследователя $u(t)$, удовлетворяющие условиям игры Γ'_C, такие, что точка $E(t+1)$, полученная применением стратегии уклонения (13), удовлетворяет условию $|E(t+1)| > R$.*

Доказательство. Пусть $\alpha = 2$, $\beta = 1$, $R = 4$, $E(t) = (4,0)$, $P(t) = (0,0)$, $u(t) = (2,0)$.

Тогда $E(t) - P(t) = (4,0)$. Следовательно, $v(t) = \beta(1,0) = (1,0)$. Отсюда $|E(t+1)| = |(4,0) + (1,0)| = |(5,0)| = 5 > 4 = R$.

□

2.3. Модификация стратегий для игры преследования в круге

Утверждения, доказанные в предыдущем разделе, показывают необходимость выработки модификаций стратегий параллельного сближения (12) и стратегии уклонения (13) для удовлетворения всем условиям из определения (19), а также модификаций, удовлетворяющих определению (20) дискретного аналога игры преследования в круге.

2.3.1. Параллельное сближение в игре Γ_C

Идея метода параллельного сближения состоит в том, что преследователь, руководствуясь целью догнать убегающего за наименьшее возможное время, в каждый момент времени движется напрямую к предполагаемой точке встречи при допущении, что вектор скорости убегающего будет оставаться

неизменным. Однако в случаях, когда стратегия параллельного сближения предписывает преследователю выйти за пределы круга C, соответствующий вектор скорости преследователя направлен в сторону точки, которая никогда не будет достигнута никаким из игроков при соблюдении условий (19) или (20). Поэтому при выборе стратегии преследования разумно учитывать возможности убегающего.

Наиболее простой вариант - двигаться к точке пересечения луча, исходящего из точки $E(t)$ в направлении вектора $v(t)$ и границы $\overline{C}$ круга C. Скорость при движении согласно данной стратегии в подвижной системе координат O_{ab} может быть записана следующим образом:

$$\begin{cases} u(t) = (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t)), |F(t)| \leq R, \\ u(t) = \alpha \frac{S(t)-P(t)}{|S(t)-P(t)|}, |F(t)| > R. \end{cases} \tag{21}$$

Здесь $S(t), F(t)$ определяются следующим образом:

$$\begin{cases} F(t) = \{E(t) + lv(t) \mid l > 0\} \cap \{P(t) + l * (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t)) \mid l > 0\}, \\ S(t) = \{E(t) + lv(t) \mid l > 0\} \cap \overline{C}. \end{cases} \tag{22}$$

Стратегия $u(t)$ преследователя может быть получена переходом в (21) к неподвижной системе координат O_{xy} посредством домножения слева на матрицу поворота.

Уравнение для точки $F(t)$ в верхней части системы (22) может быть упрощено следующим образом:

$$F(t) = \{E(t) + lv(t) \mid E(t) + lv(t) = P(t) + l * (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t))\}$$

$$F(t) = \{E(t) + lv(t) \mid E(t) - P(t) = l * (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t)) - lv(t)\}$$

$$F(t) = \{E(t) + lv(t) \mid E(t) - P(t) = l * ((\sqrt{\alpha^2 - v_b^2(t)}, v_b(t)) - v(t))\}$$

$$F(t) = \{E(t) + lv(t) \mid E(t) - P(t) = l * (\sqrt{\alpha^2 - v_b^2(t)} - \sqrt{\beta^2 - v_b^2(t)}, 0)\}$$

$$F(t) = \{E(t) + lv(t) \mid |E(t) - P(t)| = l * |(\sqrt{\alpha^2 - v_b^2(t)} - \sqrt{\beta^2 - v_b^2(t)}, 0)|\}$$

$$F(t) = \{E(t) + lv(t) \mid |E(t) - P(t)| = l * (\sqrt{\alpha^2 - v_b^2(t)} - \sqrt{\beta^2 - v_b^2(t)})\}$$

$$F(t) = E(t) + \frac{|E(t) - P(t)|}{\sqrt{\alpha^2 - v_b^2(t)} - \sqrt{\beta^2 - v_b^2(t)}} v(t) \tag{23}$$

Верхняя часть склейки (21) удовлетворяет $|P(t)| \leq R$ ввиду выполнения $F(t) \in C$ и выпуклости C. Аналогично, нижняя часть склейки (21), удовлетворяет $|P(t)| \leq R$ ввиду выполнения $S(t) \in C$ по построению и выпуклости C. Очевидно, обе части склейки (21) также удовлетворяют условию $|u(t)| \leq \alpha$. Таким образом, (21) описывает допустимую стратегию игрока П в игре (19).

2.3.2. Уклонение в игре Γ_C

Как следует из определения внутренних точек множества, любое движение убегающего E(t) будет удовлетворять $|E(t)| <= R$ на $t \in [0; T]$ при $|E(0)| < R$ и достаточно малых T. Для убегающего, находящегося на границе круга C, естественно продолжать движение по границе окружности. Данное движение соответствует вектору скорости $v(t)$, в каждый момент времени параллельному касательной к окружности $\overline{C}$ в точке $E(t)$. Если дополнительно потребовать выполнения $|v(t)| = \beta$ (что имеет смысл как требование для игрока У играть "в полную силу"), множество рассматриваемых скоростей в конкретный момент времени t сокращается до двух противонаправленных векторов, по свойству касательной перпендикулярных радиусу окружности $\overline{C}$, опущенному к точке $E(t)$.

Среди данной пары направлений движения разумно выбирать то, что направлено "от" текущего местоположения преследователя. При этом в качестве меры качества движения можно взять косинус угла между векторами

$v(t)$ и $E(t) - P(t)$. Тогда стратегия убегающего примет следующий вид:

$$\begin{cases} v(t) = K_1(t), |E(t)| = R \wedge \frac{(E(t)-P(t))*K_1(t)}{|E(t)-P(t)|*|K_1(t)|} \leq \frac{(E(t)-P(t))*K_2(t)}{|E(t)-P(t)|*|K_2(t)|}, \\ v(t) = K_2(t), |E(t)| = R \wedge \frac{(E(t)-P(t))*K_1(t)}{|E(t)-P(t)|*|K_1(t)|} > \frac{(E(t)-P(t))*K_2(t)}{|E(t)-P(t)|*|K_2(t)|}, \\ v(t) = \beta \frac{E(t)-P(t)}{|E(t)-P(t)|}, |E(t)| < R, \end{cases} \tag{24}$$

где $K_1(t), K_2(t)$ при $|E(t)| = R$ - перпендикулярные к $E(t)$ векторы длины β, определяющиеся по формулам:

$$\begin{cases} K_1(t) = \beta \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \frac{E(t)}{|E(t)|} = \frac{\beta}{R}(-y_E(t), x_E(t))^T, \\ K_2(t) = \beta \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \frac{E(t)}{|E(t)|} = \frac{\beta}{R}(y_E(t), -x_E(t))^T. \end{cases} \tag{25}$$

При этом $v(t)$, определяющаяся согласно (24)-(25), допустима по построению.

2.3.3. Параллельное сближение в игре Γ'_C

Чтобы построить допустимую модификацию стратегии параллельного сближения (16) в дискретной игре преследования (20), используем принцип, аналогичный примененному для игры (19): в случае выхода точки $F(t)$ за пределы круга C будем строить вектор скорости преследователя $u(t)$ таким образом, чтобы он был сонаправлен с $S'(t) - P(t)$ и имел максимально возможную в условиях данной игры длину. Однако, для определения точки S'(t) будем учитывать ограничение $|v(t)| = \beta$, которое в некоторых случаях не позволяет игроку У добраться до границы круга C, не меняя направление вектора $v(t)$. При этом при достижимости точки $E(t+1) = E(t) + v(t)$ за один шаг все так же будем переключаться на стратегию движения к точке

$E(t+1)$:

$$\begin{cases} u(t) = E(t) + v(t) - P(t),\ |E(t) + v(t) - P(t)| \le \alpha, \\ u(t) = (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t)),\ |E(t) + v(t) - P(t)| > \alpha \wedge |F(t)| \le R, \\ u(t) = \alpha \frac{S'(t) - P(t)}{|S'(t) - P(t)|},\ |E(t) + v(t) - P(t)| > \alpha \wedge |F(t)| > R, \end{cases} \tag{26}$$

где $F(t)$ определяется из уравнения (23) и:

$$S'(t) = E(t) + l_0 v(t),\ l_0 = \max_{E(t)+lv(t) \in C} l. \tag{27}$$

При этом, поскольку из определения $S(t)$ следует $S'(t) \in [E(t)S(t)] \subset C$, уравнение (27) может быть упрощено следующим образом:

$$S'(t) = E(t) + l_0 v(t),\ l_0 = \lfloor \frac{|S(t) - E(t)|}{|v(t)|} \rfloor = \lfloor \frac{|S(t) - E(t)|}{\beta} \rfloor. \tag{28}$$

Вместо условия $|F(t)| \le R$ можно также использовать следующее:

$$|P(t) + (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t))| \le R$$

В этом случае невозможность выхода за границы круга C учитывается только при непосредственной невозможности совершить движение в желанном направлении, а не сразу при выходе точки ожидаемой встречи за границы круга.

2.3.4. Уклонение в игре Γ'_C

Допустимую модификацию стратегии (13) игрока У в момент времени t будем строить как наилучший ответ на стратегию параллельного сближения (16) в смысле максимизации отношения $\tau = \frac{|F(t) - E(t)|}{\beta}$. Будем считать $v_a(t)$ переменной, по которой ведется максимизация, а $v(t)$ определять через $v_a(t)$

из условия $|v(t)| = \beta$. Заметим, что, с учетом (23),

$$\tau = \frac{|E(t) - P(t)|}{\sqrt{\alpha^2 - v_b^2(t)} - \sqrt{\beta^2 - v_b^2(t)}} = \frac{|E(t) - P(t)|}{\sqrt{\alpha^2 - \beta^2 + v_a^2(t)} - v_a(t)}$$

Тогда:

$$\frac{\partial \tau}{\partial v_a(t)} = \frac{(a_E - a_P)(1 - \frac{v_a(t)}{\sqrt{\alpha^2 - \beta^2 + v_a^2(t)}})}{(\sqrt{\alpha^2 - \beta^2 + v_a^2(t)} - v_a(t))^2}, \tag{29}$$

где $(a_E, b_E), (a_P, b_P)$ - координаты точек $E(t)$ и $P(t)$ в подвижной системе координат O_{ab}.

$$\alpha > \beta > 0 \Rightarrow \alpha^2 - \beta^2 > 0$$

$$\begin{cases} \alpha^2 - \beta^2 > 0, \\ v_a^2(t) \geq 0 \end{cases} \Rightarrow \sqrt{\alpha^2 - \beta^2 + v_a^2(t)} > \sqrt{v_a^2(t)} = |v_a(t)| \Rightarrow$$

$$\Rightarrow \begin{cases} \frac{v_a(t)}{\sqrt{\alpha^2 - \beta^2 + v_a^2(t)}} < 1, \\ (\sqrt{\alpha^2 - \beta^2 + v_a^2(t)} - v_a(t))^2 > 0 \end{cases} \tag{30}$$

И по построению подвижной системы координат:

$$a_E - a_P = |E(t) - P(t)| > 0 \tag{31}$$

А значит, из (29)-(31):

$$\frac{\partial \tau}{\partial v_a(t)} > 0$$

Отсюда:

$$\arg\max_{v_a(t)} \tau = \max v_a(t)$$

С учетом $E(t+1) = E(t) + v(t)$ и фиксированности $E(t)$, задачу можно переформулировать как определение точки $E(t+1)$, лежащей на окружности L с центром в $E(t)$ и радиусом β и принадлежащей кругу C, такой, что ее координата по оси O_a максимальна.

В случае, когда $Z(t) = E(t) + (\beta, 0) \in C$, таковой является $Z(t)$.

В случае, когда $Z(t) \notin C$, не все точки окружности L принадлежат замкнутому кругу C, как следствие, окружность L пересекается с окружностью $\overline{C}$, являющейся границей круга C, в двух точках: $z_1 = (a_1, b_1)$ и $z_2 = (a_2, b_2)$. В этом случае допустимыми точками являются точки дуги окружности L, заключенной между точками пересечения L и $\overline{C}$, причем, так как $Z(t) \notin C$, данная дуга откладывается в сторону, противоположную направлению оси O_a, а из этого следует, что для всех ее точек (a, b) выполняется неравенство $a \leq \min\{a_1, a_2\}$. Итого для случая $Z(t) \notin C$, решением задачи является хотя бы одна из точек z_1 и z_2.

Тогда искомая скорость $v(t)$ игрока У в подвижной системе координат имеет следующий вид:

$$\begin{cases} v(t) = (\beta, 0), \ E(t) + (\beta, 0) \in C, \\ v(t) = (\arg\max_{(a,b) \in \overline{C} \cap L} a) - E(t), \ E(t) + (\beta, 0) \notin C. \end{cases} \tag{32}$$

Уравнение окружности L при этом имеет вид:

$$(x - x_E)^2 + (y - y_E)^2 = \beta^2 \tag{33}$$

Искомая стратегия убегающего может быть получена подстановкой уравнения (33) в склейку (32) и переходом к неподвижной системе координат O_{xy} домножением слева на матрицу поворота.

Глава 3. Программная реализация

Теперь, когда введены все необходимые понятия и получены все основные формулы, можно приступать к непосредственной разработке программы, позволяющей в условиях дискретной игры Γ'_C, определяемой системой (20), строить траектории игроков для заданных стратегий и начальных условий, а также производить перебор возможных стратегий из некоторых заданных

классов при фиксированных начальных условиях.

В качестве ЯП для создания системы симуляции был выбран Haxe 4.1.5 по причине наличия как библиотек, позволяющих легко работать с графикой и взаимодействовать с пользователем, так и библиотек, предоставляющих функциональность для решения задач линейной алгебры в пространствах размерности 2.

Список использованных библиотек программного кода:

- lime 7.8.0
- openfl 9.0.2
- hxmath 0.18.0

3.1. Вывод геометрических формул

Перед тем, как приступать к реализации системы симуляции игр преследования, требуется уточнить некоторые формулы, полученные в теоретической части.

В склейке (26), описывающей стратегию параллельного сближения в игре (20), используется вектор $S'(t)$, который, в свою очередь, определяется в уравнении (28) с использованием вектора $S(t)$, который определяется в системе (22) как вектор координат точки пересечения луча, исходящего из заданной точки $E(t)$ в направлении заданного вектора $v(t)$, и заданной окружности $\overline{C}$. Поэтому для вычисления стратегии (26) игрока П требуется наличие формул для координат точки пересечения луча и окружности.

Кроме того, для определения стратегии убегающего (32) используется выбор по заданному критерию среди точек пересечения окружностей L и $\overline{C}$. Таким образом, требуется также вывод формул для координат точек пересечения двух заданных окружностей в случае их пересечения в двух точках.

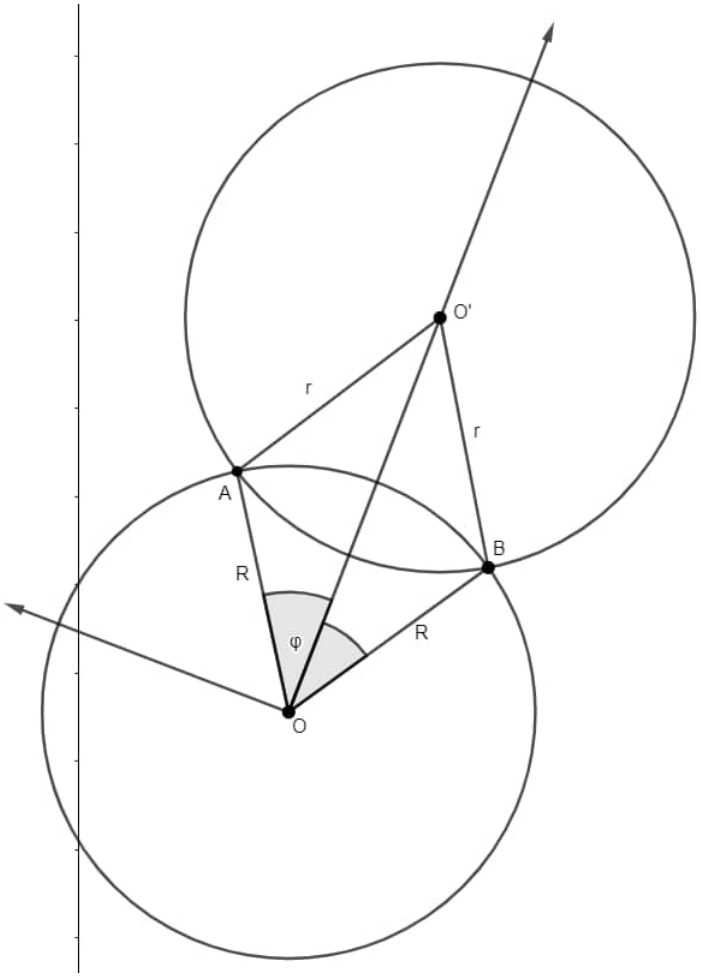

Рис. 1: Пересечение окружностей

3.1.1. Координаты пересечения окружностей

Рассмотрим окружность с центром O и радиусом R и окружность с центром O' и радиусом r такие, что $|R - r| < |OO'| < R + r$, то есть пересекающиеся в двух точках: A и B.

Введем систему координат $O_{\hat{x}\hat{y}}$, полученную из O_{xy} перенсом центра координат в точку O. Введем также систему координат $O_{\tilde{x}\tilde{y}}$, полученную из $O_{\hat{x}\hat{y}}$ поворотом на угол поворота вектора $O' - O$ относительно оси $O_{\hat{x}}$.

Данная ситуация проиллюстрирована на рис. 1

$$|OA| = |OB| = R \ \wedge \ |O'A| = |O'B| = r$$

Отсюда $\triangle OAO' = \triangle OBO'$ по трем сторонам.

Пусть тогда $\phi = \angle AOO' = \angle BOO'$. По теореме косинусов:

$$\cos\phi = \frac{R^2 + |OO'|^2 - r^2}{2R|OO'|} \tag{34}$$

Из основного тригонометрического тождества:

$$\sin\phi = \sqrt{1 - \cos^2\phi} \tag{35}$$

С помощью перехода к полярной системе координат также имеем:

$$(\tilde{x}_A, \tilde{y}_A) = (R\cos\phi, R\sin\phi) \tag{36}$$

$$(\tilde{x}_B, \tilde{y}_B) = (R\cos\phi, -R\sin\phi) \tag{37}$$

Координаты O' в системе координат $O_{\hat{x}\hat{y}}$ выражаются через исходные при помощи параллельного переноса:

$$(\hat{x}'_O, \hat{y}_{O'}) = (x_{O'} - x_O, y_{O'} - y_O) \tag{38}$$

Угол поворота $O_{\tilde{x}\tilde{y}}$ относительно $O_{\hat{x}\hat{y}}$ вычисляется по формуле:

$$\psi = atan2(\hat{y}_{O'}, \hat{x}_{O'}) \tag{39}$$

С его использованием возможно перейти обратно к $O_{\hat{x}\hat{y}}$:

$$\begin{pmatrix} \hat{x}_A \\ \hat{y}_A \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} \tilde{x}_A \\ \tilde{y}_A \end{pmatrix} \tag{40}$$

$$\begin{pmatrix} \hat{x}_B \\ \hat{y}_B \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} \tilde{x}_B \\ \tilde{y}_B \end{pmatrix} \tag{41}$$

Вернуться к O_{xy} можно также при помощи параллельного переноса:

$$(x_A, y_A) = (\hat{x}_A + x_O, \hat{y}_A + y_O) \tag{42}$$

$$(x_B, y_B) = (\hat{x}_B + x_O, \hat{y}_B + y_O) \tag{43}$$

Производя последовательно вычисления по формулам (34)-(43), можно для любых заданных окружностей вычислить координаты точек пересечения при условии, что их ровно две.

3.1.2. Координаты пересечения луча и окружности

Пусть задана окружность G радиуса R с центром в начале координат, точка A, принадлежащая кругу, описываемому данной окружностью (то есть $|OA| \leq R$), а также вектор $\vec{v}$ такой, что $A + \vec{v} \in G$. Определим координаты точки пересечения окружности G и луча, исходящего из точки A в направлении вектора v. Обозначим эту точку как S.

Данная ситуация проиллюстрирована на рис. 2

Пусть

$$\phi = \widehat{\vec{OA}, \vec{v}} \in [0; \pi] \tag{44}$$

Но так как $\vec{AS} \uparrow\uparrow \vec{v}$ по построению точки S,

$$\psi = \angle OAS = \pi - \phi \tag{45}$$

Тогда, если обозначить $x = |AS|$, по теореме косинусов:

$$x_{1,2} = |OA| \cos\psi \pm \sqrt{|OA|^2 \cos^2\psi - (|OA|^2 - R^2)} = 0 \tag{46}$$

Заметим, что аналогичные вычисления для $-\vec{v}$ дают следующее:

$$x'_{1,2} = -|OA| \cos\psi \pm \sqrt{|OA|^2 \cos^2\psi - (|OA|^2 - R^2)} = 0$$

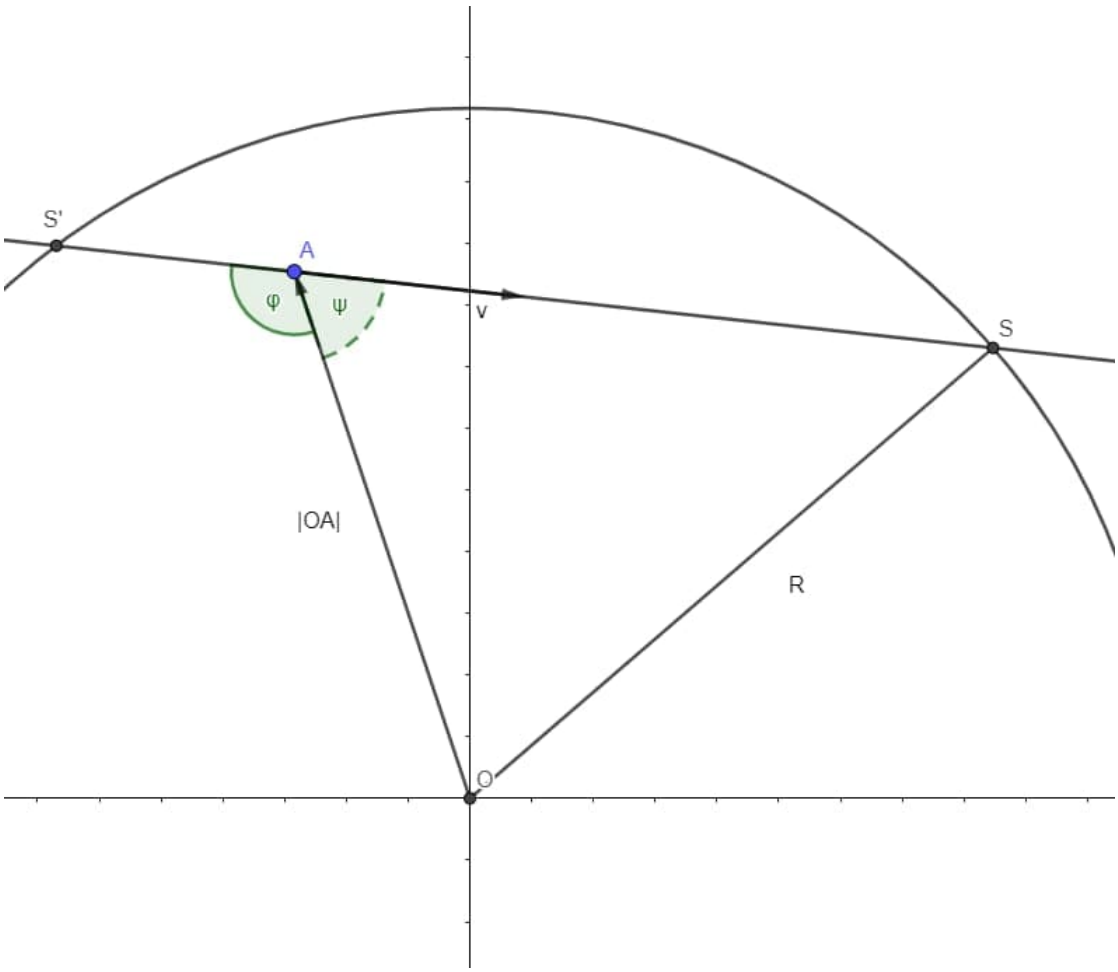

Рис. 2: Пересечение окружности и луча

Следовательно:

$$\begin{cases} x_1 = -x_2', \\ x_2 = -x_1' \end{cases}$$

Поэтому $A + x_1\frac{\vec{v}}{|\vec{v}|}$ и $A + x_2\frac{\vec{v}}{|\vec{v}|}$ - точки пересечения прямой, проходящей через точку A в направлении $\vec{v}$. Среди них лучу будут принадлежать те, для которых $x_i \geq 0$. Если A лежит на окружности G и $\vec{v}$ направлен внутрь G, таких точек будет две, причем одной из них будет A. В этом случае будем считать решением точку, не равную A. Учитывая вышесказанное, координаты искомой точки могут быть вычислены по формуле:

$$\begin{cases} S = A + x_1 \frac{\vec{v}}{|\vec{v}|}, x_2 < 0, \\ S = A + x_2 \frac{\vec{v}}{|\vec{v}|}, x_1 < 0, \\ S = A + x_1 \frac{\vec{v}}{|\vec{v}|}, x_1 \geq 0 \wedge x_2 = 0, \\ S = A + x_2 \frac{\vec{v}}{|\vec{v}|}, x_2 \geq 0 \wedge x_1 = 0, \end{cases} \tag{47}$$

где x_1, x_2 определяются из системы (44)-(46).

Склейка (47) описывает все возможные ситуации в рамках данной задачи, поскольку из условия $|OA| \leq R$ следует, что x_1 и x_2 не могут быть одновременно строго больше или строго меньше 0. Наличие ровно двух решений при этом гарантируется условием $A + \vec{v} \in G$ и выпуклостью окружности.

Решение задачи с центром окружности G не в начале координат можно получить из решения (44)-(47) при помощи параллельного переноса.

3.1.3. Реализация геометрических расчетов

Статические методы, реализующие описанные выше подсчеты, помещены в отдельный класс Geom.

Метод circleIntersections принимает на вход четыре параметра - центры окружностей center1 и center2 и их радиусы r1 и r2 соответственно. Расчеты в теле метода реализуют последовательное вычисление выражений (34)-(43). Из-за особенностей библиотеки hxmath, элементы в конструкторе матрицы поворота перечислены в порядке слева-направо сверху-вниз. Метод circleIntersections возвращает массив из двух векторов - радиус-векторов точек A и B пересечения окружностей.

Метод rayCircleIntersection принимает на вход три параметра - направляющий вектор луча dir, начало луча start и радиус окружности radius. Как и в теоретической части, рассматривается окружность с центром в начале координат. Тело метода состоит из последовательных вычислений выражений (44)-(47). Для лаконичности и быстроты подсчета, части выражения (46) слева и справа от знака плюс-минус вычисляются по отдельности, записываясь в

переменные addend1 и addend2. Для вычисления правой части используется уже посчитанное в качестве левой части выражение $|OA| \cos \phi$.

Помимо этого, для удобства тестирования системы симуляции в разных начальных условиях, в класс Geom был также добавлен статический метод randomPointInsideCircle, принимающий на вход два параметра - радиус-вектор центра center и радиус r окружности, и возвращающий случайную точку, находящуюся в круге, оганичиваемом данной окружностью. Для этого случайным образом вычисляется значение $x \in [x_{center} - r; x_{center} + r]$, затем случайным образом вычисляется значение $y \in [y_{center} - \sqrt{r^2 - x^2}; y_{center} + \sqrt{r^2 - x^2}]$. После этого с равными вероятностями выбирается одна из точек (x, y) и (y, x).

3.2. Программная реализация

3.2.1. Общая структура кодовой базы

Точка входа в программу находится в методе-конструкторе new() класса Main. В Main также определяются публичные константы: A - α, B - β, R - радиус круга C, в котором проходит игра.

Для работы с подвижной системой координат используется класс MovingCS. Для того, чтобы изменить или установить систему координат O_{ab} используется его метод set(), принимающий на вход радиус-векторы местоположений игроков У и П в системе координат O_{xy} ePos и pPos соответственно. Данный метод вычисляет угол поворота подвижной СК относительно неподвижной, а также кэширует его синус и косинус для более быстрых вычислений в дальнейшем. После установки подвижной системы координат можно использовать методы toRelative() и toAbsolute(), чтобы переводить вектор-аргумент из неподвижной СК в подвижную и в обратную сторону. Так как, согласно плану, в каждый момент времени производится только одна симуляция игры преследования, все методы и переменные класса MovingCS были сделаны статическими.

Инициализация графического интерфейса производится в методе initGraphic()

класса Main. Он принимает на вход параметры eStrat и pStrat, являющиеся стратегиями убегающего и преследователя соответственно. Стратегия убегающего определяется как фукнция, принимающая на вход местоположения игроков в системе координат O_{ab} и возвращающая вектор скорости в той же СК. Стратегия преследователя определяется аналогичным образом, но дополнительно принимает на вход вектор скорости убегающего $v(t)$. Различные стратегии игроков У и П предоставляются классами EvaderStrats и PursuerStrats соответственно в качестве статических методов.

Помимо стратегий игроков, метод initGraphic() также принимает на вход параметр mode, принимающий значения из перечислимого множества Absolute, Relative, Prescripted. Значение Relative определяет поведение по умолчанию. Значение Absolute позволяет в качестве стратегий игроков использовать функции, принимающие на вход и возвращающие вектора в неподвижной системе координат O_{xy}. Значение Prescripted используется, чтобы в качестве стратегии убегающего использовать массив вычисленных заранее скоростей $v(0), v(1), ...$ в системе координат O_{xy}, передаваемый как опциональный параметр prescriptedV. Это поведение используется для демонстрации результатов проведенной ранее симуляции с теми же начальными условиями, в частности, если был произведен поиск оптимальной стратегии убегающего на заданном классе стратегий.

Как опциональные параметры метода initGraphic() можно также задать начальные местоположения убегающего startEPos и преследователя startPPos; в случае, если эти параметры опущены, будет произведен выбор двух случайных точек как начальных условий.

Для периодического обновления позиций игроков, отображаемых на экране, метод initGraphic() обращается к статическим методам класса Demonstration. Методы класса Demonstration, в свою очередь, обращаются к методам класса Drawer для создания графических элементов: границ игровой области, точек, соответствующих игрокам, а также их траекторий.

Помимо этого используется класс Simulation, в котором содержатся ал-

горитмы поиска оптимальной стратегии на заданных классах, а также вспомогательный класс LikelihoodEvasion, методы которого используются в одном из таких алгоритмов

3.2.2. Отрисовка содержимого экрана

Класс Drawer содержит три метода - drawPoint(), drawTraj() и drawBG(), возвращающие соответствующий графический объект типа Sprite. drawBG() не принимает параметров и возвращает окружность радиуса R - границы игровой зоны. drawPoint() принимает на вход параметр type, определяющий тип точки - местоположение игрока У, местоположение игрока П или фокус $F(t)$, строящийся в других местах программы соответствии с формулой (23), а возвращает точку соответствующего цвета. drawTraj() возвращает подготовленный для отрисовки траектории объект, принимая на вход параметр тип траектории type - траектория преследователя или убегающего, в соответствии с которым определяется цвет линии, а также начальное местоположение соответствующего игрока start и координаты содержащего точку игрока контейнера bg, на основании которых выполняется расчет начальной позиции пера, из которой будет начато построение траектории. При этом и drawPoint(), и drawTraj() также добавляют возвращаемый объект в контейнер, определяемый параметром addOnto.

Экземпляр класса Demonstration хранит в себе все переменные, определяющие текущее состояние игры: местоположения игроков evaderPos и pursuerPos, время t time, местоположение фокуса $F(t)$ focalPos, а также ссылки на все элементы содержимого экрана, кроме неизменных границ игрового поля.

В классе Demonstration для каждого значения параметра mode метода initGraphic() класса Main содержится соответствующий метод: tickRelative(), tickAbsolute(), tickPrescripted(). Все эти методы работают по одному принципу: вычисляют значения скоростей игроков $v(t)$ и $u(t)$ в соответствии с их стратегиями, передаваемыми как параметры, затем обновляют переменные, описывающие состояние игры, а после обращаются к методу класса Demonstration update(), который перемещает точки игроков и точку фоку-

са и достраивает звенья траекторий игроков в соответствии с обновленными переменными состояния. Метод update() при этом также проверяет условие завершения игры (поимку убегающего) и в случае его выполнения, останавливает таймер.

Для каждого значения mode в классе Demonstration также имеются методы инициализации initRel(), initAbs(), initPresc(), создающие и запускающие таймер с периодом срабатывания в 100 милисекунд, устанавливая в качестве функции срабатывания соответствующий метод tickRelative(), tickAbsolute() или tickPrescripted() со стратегиями, передаваемыми в качестве параметров, полученными как параметры самого метода инициализации.

Метод-конструктор класса Demonstration устанавливает начальные местоположения игроков в соответствии с переданными параметрами.

Метод initGraphic() класса Main сохраняет начальные местоположения игроков в соответствии с опциональными параметрами startEPos, startPPos, либо устанавливает их случайным образом, пользуясь соответствующим методом класса Geom, в случае, если они не указаны, после чего создает объект класса Demonstration с заданными начальными условиями. Затем производится создание всех необходимых графических объектов при помощи класса Drawer. После этого производится ожидание ввода пользователя. При нажатии клавишы Enter, вызывается метод инициализации класса Demonstration, соответствующий параметру mode, который и запускает симуляцию.

3.2.3. Основные стратегии игроков

Класс EvaderStrats содержит фукнции, определяющие стратегии игрока У; класс PursuerStrats - стратегии игрока П.

Стратегия уклонения (32)-(33), определяется методом simple() класса EvaderStrats, производящим проверку условия склейки и на основнии результатов производящим соответствующие вычисления. При этом в случае, соответствующем второму уравнению склейки (32), производится делегация необходимых вычислений методу onBorder(). Это позволяет использовать его для

сокращения вычислений в тех случаях, когда принадлежность местоположения игрока У окружности $\overline{C}$ известна из специфики задачи.

Помимо этого, реализован также метод onBorderStubborn(), аналогичный onBorder(), но диктующий изменение направления движения по окружности только в случае, когда при продолжении следования текущему направлению, следующий ход неизбежно приведет к поимке убегающего. При этом направление движения сохраняется в статическую переменную класса EvaderStrats. Это - еще один аспект программы, обеспечивающий верные результаты засчет истинности утверждения о единственности исполняемой симуляции в каждый момент времени, а также единственности убегающего. Таким образом, реализована также стратегия стратегия уклонения, тяготеющая к выбранному в прошлом направлению движения.

Стратегия преследования по погонной линии (15) реализована в методе simple() класса PursuerStrats. Стратегия параллельного сближения (26), (28) реализована в методе parallel() класса PursuerStrats. Двойственность условий склейки $|F(t)| \leq R$ и $|P(t) + (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t))| \leq R$ разрешается в дальнейшем при помощи статистического эксперимента.

Глава 4. Численные эксперименты

Теперь, при наличии готовой системы симуляции, проведем эксперименты для выявления эффективных стратегий игроков. Здесь и далее, если не указано обратное, будем считать, что $\alpha = 4$, $\beta = 3.5$, $R = 250$.

4.1. Статистические эксперименты

Для начала проверим некоторые гипотезы проведением измерений результатов симуляций, а затем анализом их общепринятыми статистическими методами.

4.1.1. Уклонение с шумом

В проводимых для измерения эффективности стратегий преследователя симуляциях будем считать, что убегающий движется согласно случайным

стратегиям, близким к (32)-(33). Таковые можно получить, если на каждом шаге вычислять скорость убегающего из уравнений (32)-(33), а затем поворачивать полученный вектор скорости на случайный угол. Чтобы стратегии были близки к обозначенной, ограничим множество возможных значений угла поворота до $(-\frac{\pi}{3}, \frac{\pi}{3})$, а сам угол будем извлекать из генеральной совокупности, подчиняющейся симметричному треугольному распределению с нулевой модой и размахом $\frac{2\pi}{3}$.

Поскольку в распоряжении имется только генератор случайных чисел, распределенных равномерно на интервале $[0, 1]$, потребуется сделать преобразование, приводящее случайную величину к требуемому виду. Согласно [4], данное преобразование для треугольного распределения на интервале (a, b) с модой m имеет вид:

$$\begin{cases} a + \sqrt{y(m-a)(b-a)},\ 0 \le y \le \frac{m-a}{b-a}, \\ b - \sqrt{(1-y)(b-m)(b-a)},\ \frac{m-a}{b-a} \le y \le 1. \end{cases} \tag{48}$$

С другой стороны, по договоренности:

$$\begin{cases} a = -\frac{\pi}{3}, \\ b = \frac{\pi}{3}, \\ m = 0. \end{cases} \tag{49}$$

Поэтому, подставляя (49) в (48), получаем итоговое преобразование:

$$\phi = \frac{\pi}{3} * \begin{cases} \sqrt{2y} - 1,\ 0 \le y \le 0.5, \\ 1 - \sqrt{2(1-y)},\ 0.5 \le y \le 1. \end{cases} \tag{50}$$

Для стратегии уклонения с шумом в классе EvaderStrats был создан отдельный метод withNoise(). Поскольку случайный поворот вектора скорости близко к границе может привести к выходу за границы игровой зоны,

применение случайного поворота производится только в случае, когда длина вектора $E(t)$ не превышает $R-\beta$. В таких случаях производится вычисление скорости убегающего при помощи метода simple(), затем получение случайной величины встроенными средствами ЯП, преобразование ее согласно формуле (50), после этого поворот вектора скорости убегающего на угол, равный полученной величине. В противном случае результатом является просто возвращаемое значение метода simple().

4.1.2. Сравнение условий в стратегии параллельного сближения

Сравним эффективность стратегий параллельного сближения (26), (28) с условиями

$$|F(t)| \leq R \tag{51}$$

и

$$|P(t) + (\sqrt{\alpha^2 - v_b^2(t)}, v_b(t))| \leq R. \tag{52}$$

Для этого для разных наборов начальных условий проведем симуляции с использованием каждой из этих двух стратегий в качестве ответа на стратегию уклонения с шумом, замеряя число шагов, требуемых для поимки убегающего, а затем составим ряды из полученных наблюдений и проверим гипотезу об отсутствии эффекта обработки.

Множество комбинаций начальных позиций игроков составим следующим образом. Введем полярную систему координат с центром в начале системы координат O_{xy} и нулевым лучом, совпадающим с осью O_x. Поскольку круг C симметричен относительно начала координат, будем считать, что полярный угол начального местоположения преследователя равен 0, а радиус равен $25k_1$, где k_1 - некоторая константа. Аналогично, радиус начального местоположения убегающего положим равным $25k_2$, а полярный угол - равным $\frac{l\pi}{4}$. Теперь, варьируя k_1 и k_2 от 1 до 9 включительно, а l от 0 до 7 включительно, и при условии, что все три параметра - целые числа, получим набор из комбинаций начальных позиций игроков, на котором будем проводить эксперимент, исключая комбинации, в которых начальные местоположения

игроков совпадают. Объем выборки при этом получается равным 504.

Программная реализация модификации стратегии (26), (28) с использованием условия (52) представлена в виде метода parallel2() класса PursuerStrats. Эксперимент проводился последовательной симуляцией погони с использованием методов parallel() и parallel2(), реализующих сравниваемые стратегии, во вложенном цикле по указанным значениям параметров k_1, k_2 и l. Данная функциональность реализована в методе parallelConditionComparison() класса Simulation.

Полученные данные представлены в таблице 1 приложения. В первом столбце расположено число шагов, нужных для поимки убегающего в случае использования преследователем условия (52) в стратегии (26), (28), во втором - в случае использования условия (51).

Для проверки гипотезы об отсутствии эффекта обработки, используем критерий Пейджа для упорядоченных альтернатив. Каждую пару наблюдений для одинаковых начальных условий будем считать отдельным блоком. В таком случае модель для наблюдений можно сформулировать следующим образом:

$$x_{ij} = \mu + \tau_j + \beta_i + \epsilon_{ij},$$

где $i = \overline{1, 504}$ - номер конкретной комбинации начальных условий; $j \in \{1; 2\}$ равен единице, если наблюдение относится к стратегии с условием (51), и двойке, если наблюдение относится к стратегии с условием (52); β_i - эффект i-го блока; τ_j - эффект j-ой обработки; μ - неизвестное общее среднее; ϵ_{ij} - случайные ошибки.

Зададимся уровнем значимости $\alpha = 0.01$ и сформулируем нулевую и альтернативную гипотезу следующим образом:

$$H_0 : \tau_1 = \tau_2,$$

$$H_1 : \tau_1 < \tau_2,$$

Согласно [1], статистика Пейджа вычисляется по формуле:

$$L = \sum_{j=1}^{k} j \sum_{i=1}^{n} r_{ij}, \tag{53}$$

где k - число обработок, n - число блоков, r_{ij} - ранг наблюдения x_{ij} среди всех наблюдений в j-том блоке.

При этом исправленное значение L-статистики для больших объемов выборок равно:

$$L^* = \frac{L - \frac{nk(k+1)^2}{4}}{\sqrt{\frac{n(k^3-k)^2}{144(k-1)}}},$$

В частности, для $k = 2$, $n = 504$, что соответствует условиям проводимого теста,

$$L^* = \frac{L - 2268}{\sqrt{126}}, \tag{54}$$

Таким образом, для полученных данных:

$$\begin{cases} L = 2506.5, \\ L^* \approx 21.24726873. \end{cases}$$

А так как квантиль стандартного нормального распределения для односторонней альтернативы с уровнем значимости 0.01 приблизительно равен 2.326347874, то, поскольку $L^* > Z_{1-\alpha}$, гипотеза об отсутствии эффекта обработки H_0 отвергается в пользу гипотезы о наличии эффекта обработки с упорядочением H_1.

Полученные результаты означают, что с уверенностью 99% можно утверждать, что применение стратегии параллельного сближения (26), (28) с использованием условия (51) эффективнее в плане времени осуществления поимки игрока У, нежели чем с использованием условия (52). Поэтому в дальнейшем оставим стратегию (26), (28) без модификаций.

4.1.3. Стратегия, основанная на отношений расстояний

Заметим, что на достаточно близком к преследователю расстоянии ρ и достаточно далеко от границы игрового пространства, убегающему невыгодно совершать поворот относительно прямой, соединяющей местоположения игроков, поскольку в подобных ситуациях игра преследования в круге становится эквивалентна игре преследования на неограниченной плоскости, так как поимка может быть совершена до приближения к границе. С другой стороны, при приближении к границе потребность в повороте возрастает, поскольку, если продолжать движение к границе, то рано или поздно придется возвращаться в направлении, близком к обратному. Данные рассуждения приводят к идее стратегии, в которой необходимый поворот определяется отношением $\frac{\rho}{d}$, где d - расстояние от убегающего до точки пересечения прямой, соединяющей местоположения убегающего и преследователя, и окружности $\overline{C}$.

Будем считать, что при максимальном угле поворота, скорость игрока У будет направлена перпендикулярно радиус-вектору местоположения убегающего, а сам поворот совершается в направлении полуплоскости, отделенной прямой, соединяющей местоположение убегающего с центром круга C, не содержащей точки местоположения преследователя. В таком случае, если ввести следующее обозначение:

$$\phi = I * \widehat{\vec{OE}, \vec{PE}}, \tag{55}$$

где I - число, равное 1 в случае, когда $\vec{OE}$ находится в правой полуплоскости относительно вектора $\vec{PE}$, и -1 в противном случае, то угол ψ, на которой будет необходимо совершить поворот против часовой стрелки, будет определяться следующим образом:

$$\psi = \delta * \frac{\pi}{2} - \phi, \tag{56}$$

где δ - некоторое вещественное число между 0 и 1, определяющее величину

поворота.

δ можно определить, например, как значение дробно-рационального выражения от отношения $\frac{\rho}{d}$:

$$\delta = 1 - \frac{1}{1 + \frac{\rho}{d}}. \tag{57}$$

При этом во избежание неопределенности в выражении для δ, а также для предупреждения выхода игрока У за границы игрового поля, будем производить переключение на стратегию (32)-(33). Данная модификация имеет смысл, поскольку основная мотивация для использования стратегии (55)-(57) - совершение поворота до критического приближения к границе круга C.

Стратегию (55)-(57) реализует метод ratio() класса EvaderStrats. Для получения выборки со сравнительными результатами применения стратегии (32)-(33) и (55)-(57) используется та же сетка начальных местоположений игроков, что и в предыдущем параграфе. Для проверки также применяется критерий Пейджа с теми же нулевой и альтернативной гипотезой при том же уровне значимости. При этом $j = 1$ соответствует стратегии (32)-(33), а $j = 2$ - стратегии (55)-(57).

Результаты вычислением с использованием симуляции приведены в таблице 2 приложения. Значения статистики, вычисленные при помощи формул (53)-(54), равны следующим величинам:

$$\begin{cases} L = 2181.5, \\ L^* \approx -7.706032475. \end{cases}$$

Отсюда, так как $L^* < Z_{1-\alpha}$, нет оснований отвергнуть гипотезу об отсутствии эффекта обработки H_0 в пользу гипотезы о наличии эффекта обработки с упорядочением H_1. Это означает, что стратегия, основанная на отношении расстояний ρ и d с вероятностью 99% не дает существенного преимущества по сравнению с обычной стратегией уклонения (32)-(33).

4.2. Перебор с отсечением

Зададимся теперь целью определить достаточно эффективные стратегии убегающего, т. е. те, которые позволят отсрочить время поимки до достаточно больших значений. При этом будем считать, что преследователь движется согласно стратегии параллельного сближения. С другой стороны, будем также рассматривать движение преследователя по погонной линии.

4.2.1. Описание алгоритма

Отметим две основные преграды к использованию полного перебора:

- $|V| = \aleph_1$;
- Способ определить время поимки в игре Γ'_C для заданных местоположений игроков неизвестен.

Влияние первого ограничение, тем не менее, возможно нивелировать за счет правильного выбора дискретного множества стратегий, по которым будет вестить перебор, если исходить из предположения о том, что для заданной стратегии $v(t)$, обеспечивающей время поимки T при использованием преследователем фиксированной стратегии $u(P(t), E(t), v(t))$, и произвольного $\epsilon > 0$ найдется число δ, такое, что любая стратегия $v'(t)$, достаточно близкая к $v(t)$ в смысле $\max_t |v(t) - v'(t)| < \delta$, обеспечивает время поимки $T' \in (T - \epsilon; T + \epsilon)$, иными словами, неизвестная зависимость T от $v(t)$ при фиксированных начальных условиях и стратегии преследователя $u(P(t), E(t), v(t))$ непрерывна по $v(t)$.

Обозначим

$$W_{n,E} = \{\beta \begin{pmatrix} \cos\frac{2\pi k_i}{n} & -\sin\frac{2\pi k_i}{n} \\ \sin\frac{2\pi k_i}{n} & \cos\frac{2\pi k_i}{n} \end{pmatrix} \frac{E}{|E|} \mid k_i \in \mathbb{N} \cup \{0\}\},$$

$$e(v_1, ..., v_n) = E_0 + v_1 + ... + v_n.$$

Будем вести полный перебор по следующему множеству:

$$V' = \{(v(0), v(1), ...) \mid \forall i \; v(i) \in W_{n,e(v(0),...,v(i-1))}\}. \tag{58}$$

При этом n в формуле (58) - параметр алгоритма полного перебора. Его смысл - количество рассматриваемых направлений движения $v(i)$ на каждом шаге при заданных $v(j),\ j = \overline{0, i-1}$. Для проведения тестов будем использовать $n = 8$.

Введем также обозначение

$$V'_k = \{(v(0), ..., v(k)) \mid \forall i \; v(i) \in W_{n,e(v(0),...,v(i-1))}\}$$

и операцию конкатенации кортежей произвольной природы:

$$(s_1, ..., s_k) \sqcup (s'_1, ..., s'_l) = (s_1, ..., s_k, s'_1, ..., s'_l)\}.$$

Алгоритм полного перебора определим следующим образом:

Шаг 0. Положим $i = 1, R = (P_0, E_0, ()), \hat{R} = \varnothing$

Шаг 1. Для каждого элемента $r = (P_r, E_r, V_r) \in R$:

Шаг 1.1. Если $P_r = E_r$, перейти к следующему элементу. В противном случае, $\hat{R} = \hat{R} \cup \{(u(P_r, E_r, v), E_r + v, V_r \sqcup (v)) \mid v \in W_{n,E_r}\}$

Шаг 2. Если $\hat{R} = \varnothing$, положить $V^* = \{V_r \mid (P_r, E_r, V_r) \in R\},\ T^* = i$. В противном случае перейти к шагу 1 с $i = i + 1, R = \hat{R}, \hat{R} = \varnothing$.

Если дополнительно внести предположение о том, что не существует P_0, E_0, для которых алгоритм никогда не завершится, оптимальной будет любая стратегия убегающего, являющаяся конкатенацией какого-либо элемента множества V^* и произвольного бесконечного кортежа, состоящего из двумерных векторов длины β. Поимка при этом будет производится за время T^*.

Тем не менее, несмотря на возможность реализации описанного алго-

ритма, он не несет в себе особого практического смысла, поскольку имеет асимптотическую сложность $O(n^a)$, где a - максимально возможное число ходов для поимки при заданных начальных условиях и фиксированной стратегии преследователя. Для приемлемых значений n, в частности, для $n = 8$, и в условиях рассматриваемой игры подобный алгоритм не сможет завершиться за разумное время для большого числа комбинаций начальных позиций игроков. Поэтому имеет смысл ввести следующую оптимизацию.

После каждой итерации m, начиная с итерации a, будем производить фиксацию $v(m-a) = v_m^*$ в множестве рассматриваемых стратегий. Выбирать v_m^* при этом разумно равным значению, максимизирующим максимальное ожидаемое время поимки среди состояний $E(m), P(m)$, полученных на m-том шаге применением стратегий с $(m-a)$-той компонентой, равной v_m^*.

Однако такой принцип отсечения невыгодных стратегий в текущих условиях невозможен ввиду неизвестности зависимости времени поимки от текущего состояния игры. Это ограничение можно обойти, используя в качестве критерия отсечения максимизацию расстояния$|E(m) - P(m)|$ между игроками на m-том шаге. Алгоритм с подобной модификацией не сможет гарантировать, что полученная стратегия будет оптимальной, однако можно ожидать, что таковая будет вполне удовлетворительной (то есть обеспечивать достаточно большое время поимки), основываясь на наблюдении, что между расстоянием между игроками и временем, за которое возможна поимка, существует некоторая положительная корреляция.

Модифицированный алгоритм будет определяться следующим образом:

Шаг 0. Положим $i = 1, R = (P_0, E_0, ()), \hat{R} = \varnothing$

Шаг 1. Для каждого элемента $r = (P_r, E_r, V_r) \in R$:

Шаг 1.1. Если $P_r = E_r$, перейти к следующему элементу. В противном случае, $\hat{R} = \hat{R} \cup \{(u(P_r, E_r, v), E_r + v, V_r \sqcup (v)) \mid v \in W_{n,E_r}\}$

Шаг 2. Если $i < a$, перейти к шагу 3. В противном случае, положить

$$(P^0, E^0, V^0) = \underset{(P^r,E^r,V^r)\in\hat{R}}{\operatorname{argmax}} |E^r - P^r|$$

$$\hat{R} = \{(P^r, E^r, V^r) \in \hat{R} \mid V^r_{i-a} = V^0_{i-a}\}$$

Шаг 3. Если $\hat{R} = \varnothing$, положить $V^* = \{V_r \mid (P_r, E_r, V_r) \in R\}$, $T^* = i$. В противном случае перейти к шагу 1 с $i = i + 1, R = \hat{R}, \hat{R} = \varnothing$.

4.2.2. Реализация и результаты

Описанный алгоритм реализован в методе finiteDepthBruteForce() класса Simulation. Параметры алгоритма задаются статическими переменными класса Main. Параметру n соответствует переменная ndir, параметру a - переменная bufferSize. Если достигается итерация, номер которой равен значению переменной depth, алгоритм завершается досрочно.

Результаты применения алгоритма для разных начальных условий при ndir = 8, bufferSize = 3, depth = 500 в случае исполнения игроком П стратегии параллельного сближения (26), (28) приведены на рис. 3. Синим цветом обозначена траектория преследователя, красным - убегающего.

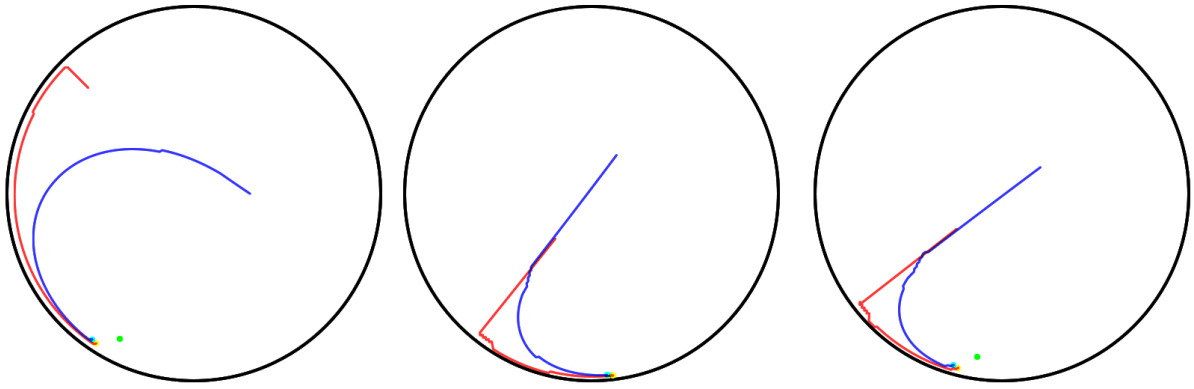

Рис. 3: Результаты применения алгоритма перебора с отсечением для стратегии параллельного сближения

Результаты применения алгоритма для разных начальных условий при ndir = 8, bufferSize = 3, depth = 500 в случае исполнения игроком П стратегии

преследования по погонной линии (15) приведены на рис. 4. Синим цветом обозначена траектория преследователя, красным - убегающего.

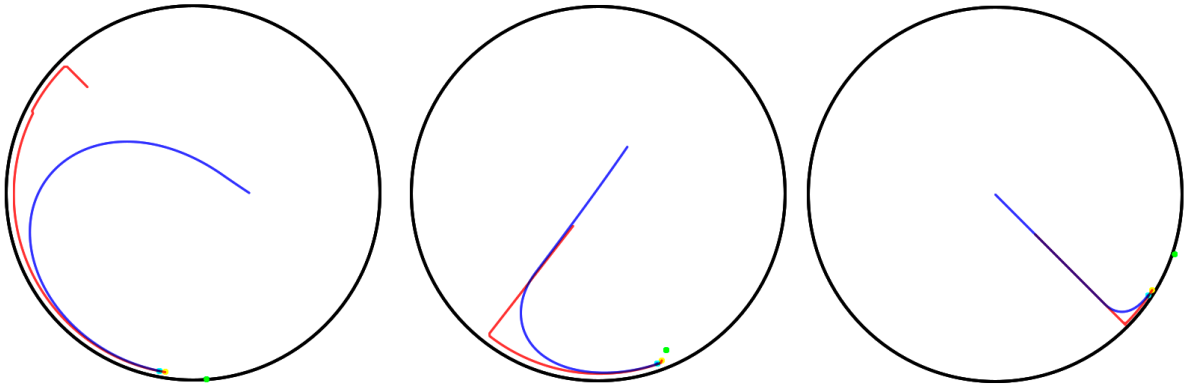

Рис. 4: Результаты применения алгоритма перебора с отсечением для стратегии преследования по погонной линии

Можно заметить, что при близких начальных условиях стратегии преследователя (15) и (26), (28) дают похожие результаты. Поэтому стратегии уклонения, полученные в данных условиях применением алгоритма перебора с отсечением, также дают близкие результаты. При их рассмотрении, выделяются их общие черты - сначала игрок У стремится добраться до границы круга, двигаясь в направлении вектора $E(t) - P(t)$, затем продолжает движение вдоль границы круга, насколько это позволяет точность, диктуемая параметром ndir.

4.3. Перебор на классе правдоподобных стратегий

Используем другой подход к поиску стратегии, обеспечивающей вполне хорошие результаты с позиции игрока У. Как уже было замечено, на достаточно близком к преследователю расстоянии и достаточно далеко от границы игрового пространства, убегающему невыгодно совершать поворот относительно прямой, соединяющей местоположения игроков, поскольку в подобных ситуациях игра преследования в круге становится эквивалентна игре преследования на неограниченной плоскости, так как поимка может быть совершена до приближения к границе. С другой стороны, весьма вероятно, выгоднее начинать поворот заранее, находясь на некотором расстояние до границы. При

этом при повороте требуется также приближаться к границе, иначе сам поворот не имеет смысла - игрок У не удаляется от центра круга C на расстояние, большее, чем некоторое a, строго меньшее радиуса круга C.

Эти соображения приводят к следующей формулировке класса трехфазовых правдоподобных стратегий убегающего с параметрами a, b и c:

1. Двигаться по прямой в направлении вектора $E(t) - P(t)$ с максимальной скоростью, пока игрок У не удалится от центра окружности C на расстояние, равное $|E(t)| < |E_0| + a(R - |E_0|)$;

2. Как только требуемое расстояние достигнуто на k-том шаге, оценить количество оставшихся шагов s в направлении вектора $E(t) - P(t)$, которые нужно сделать, чтобы достигнуть границы круга C. Положить желаемое количество шагов s' до достижения границы равным $\lfloor s(1 + \frac{b}{2}) \rfloor$. Следующие s' шагов выбирать значение $v(t)$ таким образом, чтобы скорость была максимальной, поворот вектора $E(t)$ в полуплоскость, отделяемую прямой, соединяющей центр C и местоположение игрока У, не включающую точку, соответствующую текущему местоположению игрока П, а расстояние до центра $|E^i|$ на шаге $k + i$ изменялось согласно уравнению

$$|E^i| = |E^0| + (R - |E^0|)(\frac{i}{s'})^c;$$

3. Продолжать движение по границе согласно стратегии (32)-(33).

Параметр a варьируется в пределах от 0 до 1. Его смысл заключается в том, какую долю прямолинейного пути до границы игрового поля игрок У проделает согласно стратегии уклонения в игре на неограниченной плоскости (уклонение по прямой, соединяющей текущие местоположения игроков). Параметр $b \in [0; +\infty)$ определяет, насколько долгий путь до границы будет проделан игроком У после поворота. Наконец, параметр $c \in (0; +\infty)$ определяет кривизну этого пути, при $c = 1$ траектория перемещения игрока У до границы круга C после поворота обращается в отрезок прямой.

Найти уравнение, в явной форме описывающее вторую фазу предложенной стратегии, можно при помощи применения теоремы косинусов для вычисления угла поворота вектора местоположения убегающего. Противолежащая сторона равна β из условия максимальности скорости убегающего, прилежащие стороны определяются как |E(t)| и соответствующее значение $|E^i|$ - текущий и желаемый радиусы местоположения убегающего. Найдя требуемый угол, можно найти ожидаемое местоположение на следующем шаге $E(t+1)$ домножением слева на матрицу поворота на найденный угол, а затем приведением длины данного вектора к соответствующему значению $|E^i|$ путем домножения на число $\frac{|E^i|}{|E(t)|}$. После этого скорость $v(t)$ будет определяться как $E(t+1) - E(t)$.

При критически больших или маленьких значениях параметра c, во время исполнения второй фазы приведенной выше стратегии, косинус угла поворота радиус-вектора местоположения игрока У может по модулю превысить 1. Подобные ситуации могут возникать когда приращение радиуса местоположения убегающего превышает β, потому является невозможным. В этих случаях будем считать косинус угла равным 1 для положительных значений вне допустимого промежутка и -1 - для отрицательных. Тогда действительное увеличение расстояния от местоположения убегающего до центра круга C может быть меньше ожидаемого в соответствии с описанной стратегией, однако, исходя из способа ее построения, такие ошибки будут скомпенсированы в последующих шагах.

Предложенное трехпараметрическое семейство правдоподобных стратегий убегающего реализовано в классе LikelihoodEvasion. Публичный статический метод setParams() используется для установки значений параметров семейства a, b и c, после чего метод move() может быть использован для вычисления скорости $v(t)$ игрока У для заданных местоположений игроков.

Алгоритм перебора стратегий данного семейства для разных комбинаций значений параметров реализован в методе likelyFitting() класса Simulation. Рассматриваемые значения для каждого из параметров задаются в качестве

массивов в начале метода.

Для проведения эксперимента значения параметров брались из следующих множеств:

$$\begin{cases} a \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}, \\ b \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \\ c \in \{\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{5}, \frac{5}{6}, \frac{7}{8}, 1, 1.5, 2, 2.5, 3, 3.5, 4\}. \end{cases}$$

Траектории движения игроков, при использовании преследователем стратегии параллельного сближения (26), (28) а также оптимальные значения параметров и соответствующее им время поимки, представлены на рис. 5. Синим цветом обозначена траектория преследователя, красным - убегающего.

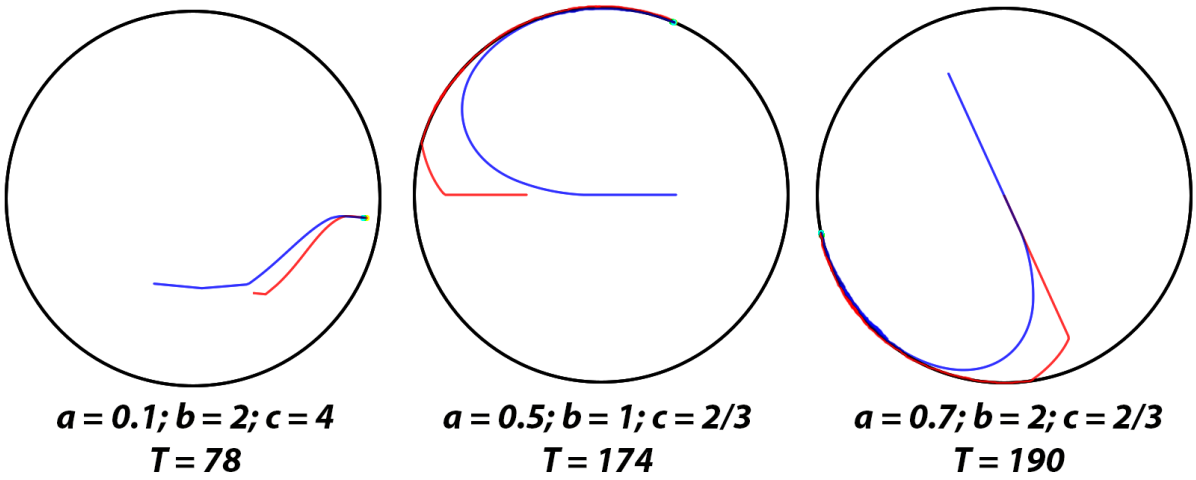

Рис. 5: Результаты применения перебора на классе правдоподобных стратегий для стратегии параллельного сближения

Траектории движения игроков, при использовании преследователем стратегии параллельного сближения (15) а также оптимальные значения параметров и соответствующее им время поимки, представлены на рис. 6. Синим цветом обозначена траектория преследователя, красным - убегающего.

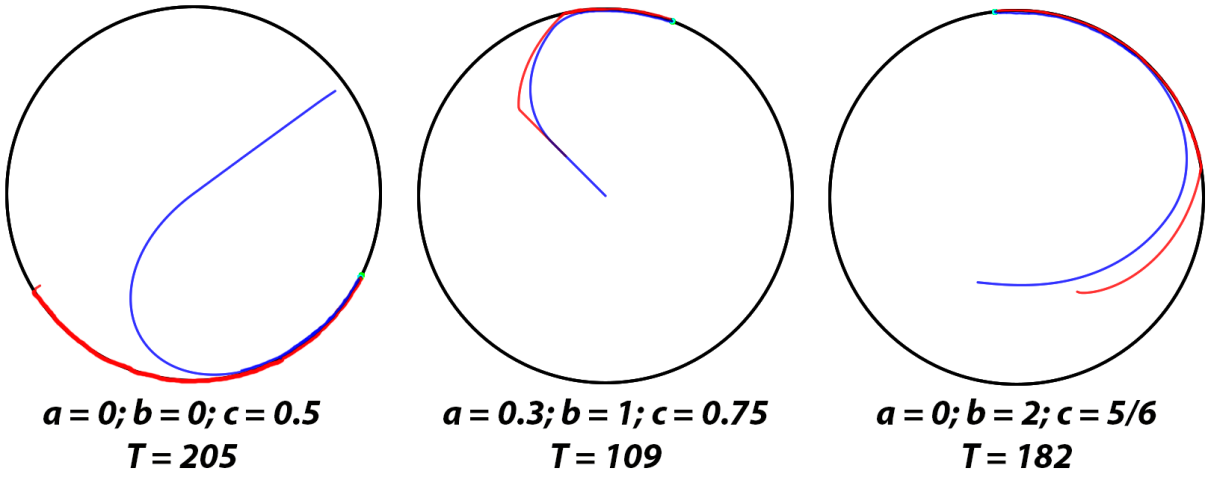

Рис. 6: Результаты применения алгоритма перебора на классе правдоподобных стратегий для стратегии преследования по погонной линии

Выводы

Несмотря на то, что результаты получены не для самой игры преследования в круге в ее исходной постановке, а для ее дискретного аналога, выводы, сделанные в ходе исследования, смогут быть использованы для доказательства более сильных утверждений касательно общей игры преследования в круге при справедливости гипотезы об аппроксимируемости ее дискретными аналогами. Эта гипотеза сделана на основе наблюдений и естественных соображений, но, в случае если данная гипотеза неверна, полученные выводы все равно будут играть важную роль для решения общей задачи после соответствующих модификаций.

Система проведения симуляций была успешно разработана и протестирована. Архитектура приложения была спроектирована в высокой мере гибкой, имеется возможность не только легко добавлять новые стратегии игроков, а также новые алгоритмы перебора, но и даже реализовывать различные смежные классы игр преследования ценой небольших изменений кодовой базы. Например, чтобы реализовать игру преследования в прочих ограниченных пространствах (эллипс, квадрат, ...), достаточно вынести условие отсечения стратегий в отдельный класс, а чтобы воплотить симуляцию игры с «линией

жизни» (17)-(18), требуется лишь дополнить условие остановки игры.

Алгоритм перебора с отсечением при верном выборе параметров и достаточном времени выполнения способен обеспечивать результаты, близкие к действительным оптимальным стратегиям. Алгоритм перебора на классе правдоподобных стратегий, несмотря на то, что производит оптимизацию на довольно узком множестве, позволяет без трудностей менять рассматриваемый класс, засчет чего становится актуальнее при увеличении количества информации о качествах оптимальных стратегий.

Таким образом, можно утверждать, что поставленная задача реализации системы симуляции дискретного аналога игры преследования в круге, была решена в полной мере. Результаты, полученные в ходе ее решения, могут стать фундаментом для дальнейших исследований в данной области.

Заключение

В ходе работы были получены следующие результаты:

- Введено определение дискретного аналога игры преследования в круге;
- Проверена допустимость ряда стратегий, значимых для игры преследования на неограниченной плоскости, а также введены допустимые аналоги для тех из них, для которых допустимость нарушается;
- Разработана система проведения симуляций игры преследования в круге с использованием различных стратегий игроков;
- Система была успешно протестирована; с ее помощью доказан ряд статистических гипотез, сравнивающих эффективность стратегий игроков;
- Были созданы два алгоритма перебора стратегий убегающего, которые впоследствии были реализованы в рамках разработанной системы симуляций;

- С использованием алгоритмов перебора были получены стратегии, оптимальные на заданных классах.

Полученные результаты позволяют с уверенностью заявить, что поставленная задача была решена в полной мере.

Список литературы

[1] Daniel W. W. et al. Applied nonparametric statistics. – 1990.

[2] Garcia E. et al. Multiple pursuer multiple evader differential games //IEEE Transactions on Automatic Control. – 2020.

[3] Karnad N., Isler V. Lion and man game in the presence of a circular obstacle //2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2009. – С. 5045-5050.

[4] Kotz S., Van Dorp J. R. Beyond beta: other continuous families of distributions with bounded support and applications. – World Scientific, 2004.

[5] Oyler D. W. Contributions To Pursuit-Evasion Game Theory : дис. – 2016.

[6] Weintraub I. E., Pachter M., Garcia E. An introduction to pursuit-evasion differential games //2020 American Control Conference (ACC). – IEEE, 2020. – С. 1049-1066.

[7] Айзекс Р. Дифференциальные игры. – Москва : Мир, 1967. – Т. 2.

[8] Зенкевич Н. А., Петросян Л. А., Семина Е. А. Теория игр //Высшая школа. – 1998.

[9] Красовский Н. Н., Субботин А. И. Позиционные дифференциальные игры. – Наука, 1974. – Т. 456.

[10] Петросян Л. А., Томский Г. В. Геометрия простого преследования. – 1983.

[11] Петросян Л. А., Томский Г. В. Через игры к творчеству. – 1991.

[12] Петросян Л. А. Дифференциальные игры преследования //Соросовский образовательный журнал. – 1995. – Т. 1. – №. 1. – С. 88-91.

Приложение

Таблицы наблюдений

Таблица 1: Сравнение условий переключения в стратегии параллельного сближения

Начало таблицы 1	
Время поимки для (52)	Время поимки для (51)
45	46
67	50
73	52
80	51
87	54
95	53
102	60
45	44
44	46
59	47
67	46
74	47
82	45
89	49
88	74
46	44
44	37
54	41
62	41
68	41
76	44
102	79

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
91	77
46	45
41	31
49	33
56	32
63	35
114	82
107	89
91	83
46	43
36	27
43	28
50	30
127	93
119	94
113	93
91	93
46	46
30	23
37	24
140	97
132	97
125	96
118	95
90	92
46	46
24	14
152	106

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
145	102
138	103
131	102
123	100
92	90
45	46
17	35
52	52
68	54
74	55
82	53
89	56
95	54
103	60
51	64
36	54
59	52
65	54
72	53
78	55
85	50
92	57
88	73
64	66
53	60
62	54
64	50
70	51

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
76	51
82	51
101	77
94	77
74	68
69	60
67	54
67	49
70	50
75	49
114	87
108	81
101	78
88	68
76	61
71	55
70	51
71	46
127	90
123	88
115	85
107	76
94	70
83	60
77	52
73	47
139	100
135	96

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
128	88
121	83
112	78
101	68
90	58
81	50
153	105
147	104
141	97
136	93
127	88
118	72
107	65
95	55
35	59
65	58
72	59
79	60
85	59
92	62
99	61
106	65
67	65
69	64
72	60
75	64
81	64
87	63

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
94	62
100	69
90	72
85	69
81	67
81	65
82	65
86	64
91	67
97	69
103	81
98	78
94	75
90	72
88	71
89	67
93	69
96	75
116	87
111	85
108	82
104	77
100	76
98	76
97	73
100	75
129	92
125	90

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
122	89
115	83
112	82
108	82
106	78
106	80
143	101
137	96
134	97
129	92
125	86
120	86
117	83
115	85
154	104
152	104
147	105
143	99
138	92
134	92
129	87
126	94
51	61
71	60
78	64
84	64
90	65
97	64

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
104	65
112	72
80	70
79	70
82	66
85	69
91	67
98	67
104	70
111	77
92	74
92	76
91	72
94	72
97	74
102	75
107	76
112	82
106	83
106	80
103	77
105	81
105	80
108	79
113	85
117	88
118	84
118	87

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
116	85
115	87
116	88
118	87
120	93
124	92
132	95
129	94
128	91
128	94
128	93
129	94
130	97
132	101
144	103
144	101
142	97
141	101
140	98
141	98
141	104
142	105
157	105
155	105
154	105
154	104
153	103
152	105

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
152	113
153	114
65	61
76	64
82	65
88	62
95	67
102	68
109	66
116	71
83	69
85	70
89	69
94	70
101	72
108	71
113	73
121	81
96	78
98	74
100	79
105	76
110	77
115	77
121	82
126	84
108	79
110	84

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
114	79
116	83
119	84
123	83
129	88
135	90
123	87
123	87
125	89
127	90
131	91
135	91
139	97
144	98
134	93
135	96
138	96
140	97
142	97
147	99
150	106
154	103
148	105
148	99
149	102
152	105
155	105
158	107

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
161	109
165	112
160	105
162	107
163	110
166	106
168	110
170	114
173	114
177	117
73	61
79	60
86	63
93	62
99	61
106	64
114	66
121	72
86	69
91	66
97	72
102	68
109	70
115	71
122	71
129	75
98	79
102	75

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
108	75
113	73
119	73
127	74
132	76
138	83
111	80
117	79
120	82
125	82
130	81
137	82
142	84
149	88
124	86
128	86
133	91
137	87
142	87
149	88
154	93
160	92
137	95
141	91
145	95
149	92
154	93
159	91

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
166	96
171	100
150	98
154	100
158	99
161	101
167	99
172	98
177	106
183	105
163	106
167	108
170	105
174	108
179	108
185	110
189	109
195	114
75	60
82	58
88	63
96	62
103	60
110	62
117	61
125	65
88	65
94	61

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
101	62
107	63
114	64
122	62
128	62
135	69
100	74
107	71
112	72
120	68
126	69
132	64
140	65
147	68
114	81
119	77
126	74
132	76
139	69
144	72
152	68
159	72
126	83
131	82
139	80
145	79
151	77
158	76

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
164	70
171	75
139	93
145	89
151	89
157	82
164	83
169	78
176	76
183	81
151	101
158	100
163	93
170	94
177	91
182	85
189	83
196	88
165	105
171	105
176	102
183	98
188	94
195	89
201	88
208	91
77	35
84	52

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
90	52
97	54
105	53
113	56
120	58
127	60
89	67
96	60
103	52
110	53
119	53
125	50
132	52
139	57
102	70
109	67
115	60
123	52
130	51
138	50
145	50
152	51
115	79
122	75
128	67
136	59
143	55
150	53

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
157	52
165	49
127	86
135	87
142	73
149	70
156	63
163	57
170	52
177	48
141	90
147	88
155	84
162	77
170	71
176	59
183	52
190	48
154	99
161	96
168	89
174	86
181	76
188	72
195	58
203	49
167	108
173	106

Продолжение таблицы 1	
Время поимки для (52)	Время поимки для (51)
181	100
188	92
195	85
202	74
208	66
215	61
Конец таблицы 1	

Таблица 2: Сравнение эффективности стратегий уклонения

Начало таблицы 2	
Время поимки для simple()	Время поимки для ratio()
49	38
54	44
54	46
54	47
54	50
54	53
59	56
49	43
49	36
49	41
49	41
49	43
49	46

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
51	49
79	65
49	44
39	34
39	36
39	37
39	39
43	42
84	71
84	70
49	44
34	31
34	31
34	32
34	36
89	77
89	76
89	74
49	45
29	26
29	27
29	29
99	85
99	82
99	81
99	79
49	46
24	21

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
24	22
104	90
104	90
104	89
104	86
99	82
49	46
14	18
109	97
109	95
109	95
109	94
109	91
99	84
49	46
19	33
57	51
55	49
54	50
54	52
54	54
54	57
61	60
57	58
39	56
54	52
49	51
47	52

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
51	52
50	54
55	57
80	67
69	65
58	60
49	56
45	52
46	53
44	54
47	55
84	72
84	72
74	68
58	64
45	58
43	55
42	53
38	56
89	80
92	80
85	78
75	71
57	64
42	60
37	57
34	56
99	87

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
96	87
96	85
88	80
72	73
57	67
40	62
33	60
104	92
105	92
99	91
97	88
87	81
70	76
51	69
39	64
109	99
109	99
107	97
103	95
99	91
88	83
69	75
46	68
38	54
58	55
54	55
58	55
57	57

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
56	58
56	61
59	64
68	60
61	60
56	59
57	60
52	60
54	62
53	65
59	68
79	67
71	68
64	66
56	66
53	67
51	68
50	70
53	71
83	74
82	73
71	73
63	73
58	72
52	72
52	75
49	75
92	81

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
87	81
83	79
73	80
66	79
55	77
52	80
49	82
96	86
94	87
91	87
82	85
75	86
64	85
57	86
48	87
101	93
103	94
95	94
92	92
82	91
72	91
62	92
55	92
111	100
106	101
102	100
99	100
89	98

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
78	99
70	99
61	98
55	55
58	57
58	56
56	59
60	61
59	63
59	66
64	68
68	62
66	62
64	63
61	65
61	67
57	69
55	72
64	74
78	67
74	69
66	70
63	72
62	73
59	76
58	78
63	81
81	74

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
81	75
73	76
72	77
68	80
64	82
60	85
65	87
90	81
86	82
82	83
78	85
72	87
68	89
63	92
68	93
94	87
92	89
89	91
84	91
78	94
73	96
68	98
67	101
104	94
100	95
98	97
90	98
83	99

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
78	103
73	105
71	108
108	100
108	102
104	103
97	105
92	107
87	110
78	112
74	114
65	56
61	56
63	56
61	59
59	61
59	63
59	66
65	68
71	61
66	62
65	63
63	66
64	67
61	70
64	72
67	75
78	67

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
75	69
72	70
70	72
66	74
65	76
65	79
71	81
86	74
83	76
79	76
72	78
70	80
70	83
68	85
72	88
89	79
89	81
86	83
80	84
78	87
75	89
75	92
76	94
99	86
96	87
90	89
85	91
85	92

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
79	96
78	98
82	100
103	93
99	94
95	95
93	97
90	99
85	102
84	105
86	107
108	98
107	100
102	101
102	104
96	105
89	108
91	110
94	112
63	53
62	53
65	54
63	57
63	58
62	60
62	63
70	66
72	60

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
71	60
69	60
65	61
67	63
65	65
69	67
73	70
80	65
74	65
74	66
72	66
72	67
73	69
73	71
80	73
83	72
80	71
82	71
77	72
80	73
74	74
81	76
84	78
93	77
87	78
87	77
85	77
85	78

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
83	79
87	81
87	82
97	84
95	84
93	83
89	83
88	83
88	84
93	86
93	87
102	91
104	90
100	89
99	89
96	89
91	89
98	91
98	91
111	97
107	96
107	95
106	94
101	94
99	95
102	95
105	96
67	51

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
64	49
63	50
62	52
62	54
67	55
66	58
71	61
69	57
69	54
71	54
69	53
68	54
72	56
71	57
74	60
78	63
76	60
77	58
73	57
76	56
74	57
80	58
80	60
82	70
84	66
83	63
84	61
81	60

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
83	59
84	60
87	61
92	75
88	72
91	69
86	66
86	64
88	62
95	62
95	63
97	82
97	78
94	75
93	71
93	68
94	66
97	65
99	65
101	89
101	84
103	81
101	77
100	73
100	70
106	68
108	67
111	95

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
110	91
107	87
104	82
108	77
109	74
116	71
112	70
68	32
67	45
67	46
66	47
66	49
66	50
66	54
72	56
72	54
71	47
70	43
69	43
69	45
74	46
74	48
76	51
77	62
80	55
79	47
78	43
77	43

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
77	42
82	44
85	46
86	69
84	63
83	55
82	48
86	43
85	41
91	41
90	43
91	76
89	71
92	63
91	54
90	47
90	42
95	40
98	40
96	82
99	78
97	70
95	61
94	53
98	46
101	40
107	38
101	89

Продолжение таблицы 2	
Время поимки для simple()	Время поимки для ratio()
104	84
101	78
105	69
103	60
109	51
111	43
109	38
111	94
108	90
111	84
109	76
108	67
117	57
115	47
120	40
Конец таблицы 2	

Листинг программного кода

Main.hx

```
package;

import Demonstration.PursuerStrategy;
import Demonstration.EvaderStrategy;
import openfl.ui.Keyboard;
import openfl.events.KeyboardEvent;
import hxmath.math.Vector2;
import openfl.display.Sprite;
```

```
enum Mode
{
    Absolute;
    Relative;
    Prescripted;
}

class Main extends Sprite
{
    public static var R:Float = 250;
    public static var A:Float = 4;
    public static var B:Float = 3.5;

    public static var ndir = 8;
    public static var depth = 500;
    public static var bufferSize = 3;

    public static var Rsq:Float = R * R;
    public static var sectorAngle:Float = 2 * Math.PI / ndir;

    public function initGraphic(eStrat:EvaderStrategy, pStrat:
        PursuerStrategy, mode:Mode, ?startEPos:Vector2, ?startPPos:
        Vector2, ?prescriptedV:Array<Vector2>)
    {
        var evaderPos = startEPos != null? startEPos : Geom.
            randomPointInsideCircle(Vector2.zero, R);
        var pursuerPos = startPPos != null? startPPos : Geom.
            randomPointInsideCircle(Vector2.zero, R);

        var demo:Demonstration = new Demonstration(evaderPos,
            pursuerPos);

        var bg:Sprite = Drawer.drawBG();
        addChild(bg);

        demo.focal = Drawer.drawPoint(Focal, bg);
```

```
        demo.evader = Drawer.drawPoint(Evader, bg);
        demo.pursuer = Drawer.drawPoint(Pursuer, bg);

        demo.evaderTraj = Drawer.drawTraj(Evader, evaderPos, bg, this)
            ;
        demo.pursuerTraj = Drawer.drawTraj(Pursuer, pursuerPos, bg,
            this);

        var onPress:KeyboardEvent->Void;

        onPress = (e) ->
        {
            if (e.keyCode != Keyboard.ENTER)
                return;

            stage.removeEventListener(KeyboardEvent.KEY_DOWN, onPress);
            switch mode
            {
                case Absolute: demo.initAbs(eStrat, pStrat);
                case Relative: demo.initRel(eStrat, pStrat);
                case Prescripted: demo.initPresc(prescriptedV, pStrat);
            }
        };

        stage.addEventListener(KeyboardEvent.KEY_DOWN, onPress);
    }

    public function new()
    {
        super();
        //For random start positions:
        var evaderPosStart = Geom.randomPointInsideCircle(Vector2.zero
            , R);
        var pursuerPosStart = Geom.randomPointInsideCircle(Vector2.
            zero, R);
        //For fixed start positions (fromPolar args can be replaced):
        /*var evaderPosStart = Vector2.fromPolar(Math.PI, 0);
```

```
        var pursuerPosStart = Vector2.fromPolar(-2, 7 * 25);*/

        //For optimization over the likely strategies (same goes for
           brute force simulation):
        var vs = Simulation.likelyFitting(PursuerStrats.parallel,
           evaderPosStart, pursuerPosStart);
        initGraphic(EvaderStrats.simple, PursuerStrats.parallel,
           Prescripted, evaderPosStart, pursuerPosStart, vs);

        //To test one of the likely strategies:
        /*LikelihoodEvasion.setParams(evaderPosStart, pursuerPosStart,
            0.1, 5, 2);
          LikelihoodEvasion.init();
        initGraphic(LikelihoodEvasion.move, PursuerStrats.parallel,
           Relative, evaderPosStart, pursuerPosStart);*/

        //Just to simulate the usage of the two fixed strategies:
        //initGraphic(EvaderStrats.simple, PursuerStrats.parallel,
           Relative, evaderPosStart, pursuerPosStart);

        //To gather experimental data (ratio() efficiency test or
           parallel() vs parallel2()):
        //Simulation.ratioTest();

    }
}
```

Demonstration.hx

```
package;

import openfl.display.Sprite;
import hxmath.math.Vector2;
import haxe.Timer;

typedef EvaderStrategy = (ePos:Vector2, pPos:Vector2) -> Vector2;
typedef PursuerStrategy = (ePos:Vector2, pPos:Vector2, v:Vector2) ->
    Vector2;
```

```
class Demonstration
{
    private var t:Timer;
    private var time:Int;
    private var evaderPos:Vector2;
    private var pursuerPos:Vector2;
    private var focalPos:Vector2;

    public var evader:Sprite;
    public var pursuer:Sprite;
    public var focal:Sprite;
    public var evaderTraj:Sprite;
    public var pursuerTraj:Sprite;

    private function tickRelative(eStrat:EvaderStrategy, pStrat:
       PursuerStrategy)
    {
        MovingCS.set(evaderPos, pursuerPos);

      var oldRelEPos = MovingCS.toRelative(evaderPos);
        var oldRelPPos = MovingCS.toRelative(pursuerPos);

        var vRel:Vector2 = eStrat(oldRelEPos, oldRelPPos);
        var uRel:Vector2 = pStrat(oldRelEPos, oldRelPPos, vRel);

        vRel = vRel.normalizeTo(Main.B);
      var newRelEPos = oldRelEPos + vRel;
        var newRelPPos = oldRelPPos + uRel;
        var relFocalPos = focalPoint(oldRelEPos, oldRelPPos, vRel,
           uRel);

      evaderPos = MovingCS.toAbsolute(newRelEPos);
        pursuerPos = MovingCS.toAbsolute(newRelPPos);
      focalPos = MovingCS.toAbsolute(relFocalPos);

        update(evaderPos, pursuerPos, focalPos);
```

```
}

private function tickAbsolute(eStrat:EvaderStrategy, pStrat:
    PursuerStrategy)
{
    var v:Vector2 = eStrat(evaderPos, pursuerPos);
    var u:Vector2 = pStrat(evaderPos, pursuerPos, v);

    evaderPos += v;
    pursuerPos += u;
    focalPos = focalPoint(evaderPos, pursuerPos, v, u);

    update(evaderPos, pursuerPos, focalPos);
}

private function tickPrescripted(eStrat:Array<Vector2>, pStrat:
    PursuerStrategy)
{
    if (time + 1 >= eStrat.length)
        t.stop();

    MovingCS.set(evaderPos, pursuerPos);

  var oldRelEPos = MovingCS.toRelative(evaderPos);
    var oldRelPPos = MovingCS.toRelative(pursuerPos);

    var v:Vector2 = eStrat[time];
    var vRel:Vector2 = MovingCS.toRelative(v);
    var uRel:Vector2 = pStrat(oldRelEPos, oldRelPPos, vRel);

    var newRelPPos = oldRelPPos + uRel;
    var relFocalPos = focalPoint(oldRelEPos, oldRelPPos, vRel,
        uRel);

  evaderPos += v;
    pursuerPos = MovingCS.toAbsolute(newRelPPos);
  focalPos = MovingCS.toAbsolute(relFocalPos);
```

```
    update(evaderPos, pursuerPos, focalPos);
}

private function update(evaderPos:Vector2, pursuerPos:Vector2,
   focalPos:Vector2)
{
    evaderTraj.graphics.lineTo(evaderPos.x, evaderPos.y);
  evader.x = evaderPos.x;
    evader.y = evaderPos.y;

  pursuerTraj.graphics.lineTo(pursuerPos.x, pursuerPos.y);
  pursuer.x = pursuerPos.x;
    pursuer.y = pursuerPos.y;

  focal.x = focalPos.x;
  focal.y = focalPos.y;

    if (evaderPos.distanceTo(pursuerPos) <= Main.A - Main.B)
        t.stop();
  time += 1;
}

public function initRel(eStrat:EvaderStrategy, pStrat:
   PursuerStrategy)
{
    t = new Timer(100);
    t.run = tickRelative.bind(eStrat, pStrat);
}

public function initAbs(eStrat:EvaderStrategy, pStrat:
   PursuerStrategy)
{
    t = new Timer(100);
    t.run = tickAbsolute.bind(eStrat, pStrat);
}
```

```
    public function initPresc(eStrat:Array<Vector2>, pStrat:
        PursuerStrategy)
    {
        t = new Timer(100);
        t.run = tickPrescripted.bind(eStrat, pStrat);
    }

    public function new(eStartPos:Vector2, pStartPos:Vector2)
    {
        time = 0;
        evaderPos = eStartPos.clone();
        pursuerPos = pStartPos.clone();
    }

    public static function focalPoint(ePos:Vector2, pPos:Vector2, v:
        Vector2, u:Vector2):Vector2
    {
        var T:Float = (ePos.x - pPos.x) / (u.x - v.x);
        return ePos + T * v;
    }
}
```

Simulation.hx

```
package;

import sys.io.File;
import Demonstration.PursuerStrategy;
import hxmath.math.Vector2;

typedef Node =
{
    var dir:Int;
    var epos:Vector2;
    var ppos:Vector2;
    var next:Null<Map<Int, Node>>;
}
```

```
class Simulation
{

    private static function calcMaxDist(node:Node):{dist:Float,
        branch:Null<Node>}
    {
        if (node.next == null)
            return {dist: (node.epos - node.ppos).lengthSq, branch:
                null};

        var currentMax:Float = Math.NEGATIVE_INFINITY;
        var currentArgmax:Null<Node> = null;
        for (branch in node.next)
        {
            var md = calcMaxDist(branch);
            if (md.dist > currentMax)
            {
                currentMax = md.dist;
                currentArgmax = branch;
            }
        }
        return {dist: currentMax, branch: currentArgmax};
    }

    private static function gatherLeaves(node:Node):Array<Node>
    {
        if (node.next == null)
            return [node];

        var a:Array<Node> = [];
        for (branch in node.next)
            a = a.concat(gatherLeaves(branch));
        return a;
    }

    //-----------------------------------------------------------------
```

```
public static function parallelConditionComparison()
{
   var s:String = "";
   for (a in 0...8)
      for (i in 1...9)
         for (j in 1...9)
         {
            if (a == 0 && i == j)
               continue;

            var epos:Vector2 = Vector2.fromPolar(a * Math.PI / 8,
                j * 25);
            var ppos:Vector2 = Vector2.fromPolar(0, i * 25);
            var t1 = 0;

                while (epos.distanceTo(ppos) > Main.A - Main.B)
                {
               MovingCS.set(epos, ppos);
               var erel = MovingCS.toRelative(epos);
                     var prel = MovingCS.toRelative(ppos);
               var vrel = EvaderStrats.withNoise(erel, prel);

                     ppos = MovingCS.toAbsolute(prel +
                        PursuerStrats.parallel2(erel, prel, vrel))
                        ;
               epos = MovingCS.toAbsolute(erel + vrel);

               t1++;
            }

            epos = Vector2.fromPolar(a * Math.PI / 4, j * 25);
            ppos = Vector2.fromPolar(0, i * 25);
            var t2 = 0;

                while (epos.distanceTo(ppos) > Main.A - Main.B)
                {
```

```
                MovingCS.set(epos, ppos);
                var erel = MovingCS.toRelative(epos);
                    var prel = MovingCS.toRelative(ppos);
                var vrel = EvaderStrats.withNoise(erel, prel);

                    ppos = MovingCS.toAbsolute(prel +
                        PursuerStrats.parallel(erel, prel, vrel));
                epos = MovingCS.toAbsolute(erel + vrel);

                t2++;
            }

            s += '$t1    $t2\n';
        }
    File.saveContent("Q:\\Github\\pursuit\\Export\\results_par.txt
        ", s);
}

public static function ratioTest()
{
    var s:String = "";
    for (a in 0...8)
        for (i in 1...9)
            for (j in 1...9)
            {
                if (a == 0 && i == j)
                    continue;

                var epos:Vector2 = Vector2.fromPolar(a * Math.PI / 8,
                    j * 25);
                var ppos:Vector2 = Vector2.fromPolar(0, i * 25);
                var t1 = 0;

                trace(a, i, j);

                    while (epos.distanceTo(ppos) > Main.A - Main.B)
                    {
```

```
                MovingCS.set(epos, ppos);
                var erel = MovingCS.toRelative(epos);
                    var prel = MovingCS.toRelative(ppos);
                var vrel = EvaderStrats.simple(erel, prel);
                vrel = vrel.normalizeTo(Main.B);

                    ppos = MovingCS.toAbsolute(prel +
                        PursuerStrats.parallel(erel, prel, vrel));
                epos = MovingCS.toAbsolute(erel + vrel);

                t1++;
            }

            epos = Vector2.fromPolar(a * Math.PI / 4, j * 25);
            ppos = Vector2.fromPolar(0, i * 25);
            var t2 = 0;

                while (epos.distanceTo(ppos) > Main.A - Main.B)
                {
                MovingCS.set(epos, ppos);
                var erel = MovingCS.toRelative(epos);
                    var prel = MovingCS.toRelative(ppos);
                var vrel = EvaderStrats.ratio(erel, prel);
                vrel = vrel.normalizeTo(Main.B);

                    ppos = MovingCS.toAbsolute(prel +
                        PursuerStrats.parallel(erel, prel, vrel));
                epos = MovingCS.toAbsolute(erel + vrel);

                t2++;
            }

            s += '$t1    $t2\n';
        }
    File.saveContent("Q:\\Github\\pursuit\\Export\\results_ratio.
        txt", s);
}
```

```
public static function finiteDepthBruteForce(pursuerStrat:
    PursuerStrategy, ?startEPos:Vector2, ?startPPos:Vector2):Array<
    Vector2>
{
    var evaderPosStart = startEPos != null? startEPos : Geom.
        randomPointInsideCircle(Vector2.zero, Main.R);
    var pursuerPosStart = startPPos != null? startPPos : Geom.
        randomPointInsideCircle(Vector2.zero, Main.R);

    var savedPath:Array<Vector2> = [];
    var kernel:Node = {epos: evaderPosStart, ppos: pursuerPosStart
        , dir: 0, next: null};
    var leaves:Array<Node> = [kernel];

    for (i in 0...Main.depth)
    {
        var newLeafCount = 0;
        for (leaf in leaves)
        {
            leaf.next = [];
            MovingCS.set(leaf.epos, leaf.ppos);
            var v = leaf.epos.clone().normalizeTo(Main.B);
            for (j in 0...Main.ndir)
            {
                var nextEPos = leaf.epos + v;
                if (nextEPos.lengthSq > Main.Rsq)
                    continue;

                var nextPPos = leaf.ppos + MovingCS.toAbsolute(
                    pursuerStrat(MovingCS.toRelative(leaf.epos),
                    MovingCS.toRelative(leaf.ppos), MovingCS.toRelative
                    (v)));
                if ((nextEPos - nextPPos).length < Main.A - Main.B)
                    continue;

                var newLeaf:Node = {ppos: nextPPos, epos: nextEPos,
```

```
                dir: j, next: null};
            leaf.next[j] = newLeaf;
            newLeafCount++;

            v.rotate(Main.sectorAngle, Vector2.zero);
        }
    }

    if (newLeafCount == 0)
        return convertToPresription(evaderPosStart, savedPath);

    if (i >= Main.bufferSize)
    {
        var mddata = calcMaxDist(kernel);
        savedPath.push(mddata.branch.epos);
        kernel = mddata.branch;
    }

    leaves = gatherLeaves(kernel);
    trace(i);
  }

  return convertToPresription(evaderPosStart, savedPath);
}

public static function likelyFitting(pursuerStrat:PursuerStrategy
  , startEPos:Vector2, startPPos:Vector2):Array<Vector2>
{
    var aValues:Array<Float> = [for (i in 0...10) i / 10];
    var bValues:Array<Int> = [for (j in 0...10) j];
    var cValues:Array<Float> = [1/2, 2/3, 3/4, 3/5, 4/5, 5/6,
      7/8, 1, 1.5, 2, 2.5, 3, 3.5, 4];

    var aOptimal:Float = 0;
    var bOptimal:Int = 0;
    var cOptimal:Float = 0;
    var optimalT:Int = 0;
```

```
var optimalVs:Array<Vector2> = [];

for (a in aValues)
    for (b in bValues)
        for (c in cValues)
        {
            LikelihoodEvasion.setParams(startEPos, startPPos
                , a, b, c);
            LikelihoodEvasion.init();

            var epos:Vector2 = startEPos.clone();
            var ppos:Vector2 = startPPos.clone();
            var vs:Array<Vector2> = [];

            while (epos.distanceTo(ppos) > Main.A - Main.B)
            {
          MovingCS.set(epos, ppos);
          var erel = MovingCS.toRelative(epos);
                var prel = MovingCS.toRelative(ppos);
          var vrel = LikelihoodEvasion.move(erel, prel);
          vrel = vrel.normalizeTo(Main.B);

          vs.push(MovingCS.toAbsolute(vrel));

          epos = MovingCS.toAbsolute(erel + vrel);
                ppos = MovingCS.toAbsolute(prel +
                   pursuerStrat(erel, prel, vrel));
            }

            if (vs.length > optimalT)
            {
                aOptimal = a;
                bOptimal = b;
                cOptimal = c;
                optimalT = vs.length;
                optimalVs = vs;
            }
```

```
                    trace(a, b, c);
                }

            trace("Optimal: ", aOptimal, bOptimal, cOptimal);
        trace("Optimal t: ", optimalT);
        return optimalVs;
    }

    private static function convertToPresription(startPos:Vector2,
        bruteResult:Array<Vector2>):Array<Vector2>
    {
        var result:Array<Vector2> = [];
        var pos:Vector2 = startPos.clone();

        for (epos in bruteResult)
        {
            var v = epos - pos;
            result.push(v);
            pos = epos;
        }

        return result;
    }
}
```

LikelihoodEvasion.hx

```
package;

import hxmath.math.Vector2;

enum Phase
{
    Line;
    Approaching;
    Free;
}
```

```
class LikelihoodEvasion
{
    public static var turnRadius:Float;
    public static var turnRadiusSq:Float;
    public static var b:Int;
    public static var c:Float;

    public static var phase:Phase;

    public static var minimumSteps:Int;
    public static var totalSteps:Int;
    public static var actualTurningRadius:Float;
    public static var rotationDir:Int;

    public static var currentStep:Int;


    public static function calcDistToBorder(ePos:Vector2, pPos:
       Vector2):Float
    {
        var E = ePos.length;
        var dr = ePos - pPos;
        var cosphi = -(ePos * dr / (E * dr.length));
        return E * cosphi + Math.sqrt(E * E * (cosphi * cosphi - 1)
           + Main.R * Main.R);
    }

    public static function setParams(ePos:Vector2, pPos:Vector2, a:
       Float, b:Int, c:Float)
    {
        turnRadius = a * Main.R + (1 - a) * ePos.length;
        turnRadiusSq = turnRadius * turnRadius;
        LikelihoodEvasion.b = b;
        LikelihoodEvasion.c = c;
    }

    public static function init()
```

```
{
    phase = Line;
}

public static function move(ePos:Vector2, pPos:Vector2):Vector2
{
    switch phase
    {
        case Line:
            if (ePos.lengthSq < turnRadiusSq)
                return new Vector2(Main.B, 0);
            else
            {
                var dist = calcDistToBorder(ePos, pPos);
                minimumSteps = Math.ceil(dist / Main.B);
                totalSteps = minimumSteps + b * Math.floor(
                   minimumSteps / 2);
                actualTurningRadius = ePos.length;
                rotationDir = ePos.signedAngleWith(pPos) > 0? -1
                    : 1;
                currentStep = 0;

                phase = Approaching;
                return move(ePos, pPos);
            }
        case Approaching:
            var oldRadius = ePos.length;
            if (currentStep == totalSteps)
            {
                phase = Free;
                if (oldRadius < Main.R - Main.B)
                    return ePos.normalizeTo(Main.B);
                else
                    return move(ePos, pPos);
            }
            currentStep++;
```

```
                var newRadius = actualTurningRadius + (Main.R -
                    actualTurningRadius) * Math.pow(currentStep /
                    totalSteps, c);
                var newPos = ePos.clone();
                newPos.normalizeTo(newRadius);
                var cosBetween = (newRadius * newRadius + oldRadius
                    * oldRadius - Main.B * Main.B)/(2 * newRadius *
                    oldRadius);
                var angleBetween = cosBetween > 1? 0 : cosBetween <
                    -1? Math.PI : Math.acos(cosBetween);
                newPos.rotate(rotationDir * angleBetween, Vector2.
                    zero);

                return (newPos - ePos).normalizeTo(Main.B);
            case Free:
                return EvaderStrats.onBorder(ePos, pPos);
        }
    }
}
```

PursuerStrats.hx

```
package;

import hxmath.math.Vector2;

class PursuerStrats
{

    public static var lastDir:Null<Vector2>;
    public static var toggled:Bool = false;

    public static function simple(ePosRel:Vector2, pPosRel:Vector2,
        vRel:Vector2):Vector2
    {
        var dr = ePosRel + vRel - pPosRel;
        if (dr.lengthSq <= Main.A * Main.A)
            return dr;
```

```
        return dr.normalizeTo(Main.A);
    }

    public static function parallel(ePosRel:Vector2, pPosRel:Vector2
        , vRel:Vector2):Vector2
    {
        var dr = ePosRel + vRel - pPosRel;
        if (dr.lengthSq <= Main.A * Main.A)
            return dr;

        var uRelXSq = Main.A * Main.A - vRel.y * vRel.y;
        var uRel = new Vector2(Math.sqrt(uRelXSq), vRel.y);

        if (Demonstration.focalPoint(ePosRel, pPosRel, vRel, uRel).
            lengthSq <= Main.Rsq)
            return uRel;
        else
        {
            var S = Geom.rayCircleIntersection(vRel, ePosRel, Main.R
                );
            var l0 = Math.floor(S.distanceTo(ePosRel) / Main.B);
            var S1 = ePosRel + l0 * vRel;
            return (S1 - pPosRel).normalizeTo(Main.A);
        }
    }

    public static function parallel2(ePosRel:Vector2, pPosRel:
        Vector2, vRel:Vector2):Vector2
    {
        var dr = ePosRel + vRel - pPosRel;
        if (dr.lengthSq <= Main.A * Main.A)
            return dr;

        var uRelXSq = Main.A * Main.A - vRel.y * vRel.y;
        var uRel = new Vector2(Math.sqrt(uRelXSq), vRel.y);
```

```
        if ((pPosRel + uRel).lengthSq <= Main.Rsq)
            return uRel;
        else
        {
            var S = Geom.rayCircleIntersection(vRel, ePosRel, Main.R
                );
            var l0 = Math.floor(S.distanceTo(ePosRel) / Main.B);
            var S1 = ePosRel + l0 * vRel;
            return (S1 - pPosRel).normalizeTo(Main.A);
        }
    }

    public static function toggle(ePosRel:Vector2, pPosRel:Vector2,
       vRel:Vector2):Vector2
    {
        var dr = ePosRel + vRel - pPosRel;
        if (dr.lengthSq <= Main.A * Main.A)
            return dr;

        var uRelXSq = Main.A * Main.A - vRel.y * vRel.y;
        var uRel = new Vector2(Math.sqrt(uRelXSq), vRel.y);

        if ((pPosRel + uRel).lengthSq <= Main.Rsq/*Demonstration.
           focalPoint(ePosRel, pPosRel, vRel, uRel).lengthSq <= Main.
           Rsq*/)
        {
            if (lastDir != null && Math.abs(lastDir.signedAngleWith(
               uRel)) > 0.5 * Math.PI)
                uRel = (ePosRel - pPosRel).normalizeTo(Main.A);
            lastDir = uRel.clone();
            return uRel;
        }
        else
        {
            var S = Geom.rayCircleIntersection(vRel, ePosRel, Main.R
                );
            var l0 = Math.floor(S.distanceTo(ePosRel) / Main.B);
```

```
            var S1 = ePosRel + l0 * vRel;
            return (S1 - pPosRel).normalizeTo(Main.A);
        }
    }

    public static function oneTimeToggle(ePosRel:Vector2, pPosRel:
        Vector2, vRel:Vector2):Vector2
    {
        if (toggled)
            return simple(ePosRel, pPosRel, vRel);

        var dr = ePosRel + vRel - pPosRel;
        if (dr.lengthSq <= Main.A * Main.A)
            return dr;

        var uRelXSq = Main.A * Main.A - vRel.y * vRel.y;
        var uRel = new Vector2(Math.sqrt(uRelXSq), vRel.y);

        if ((pPosRel + uRel).lengthSq <= Main.Rsq/*Demonstration.
            focalPoint(ePosRel, pPosRel, vRel, uRel).lengthSq <= Main.
            Rsq*/)
        {
            if (lastDir != null && Math.abs(lastDir.signedAngleWith(
                uRel)) > 0.5 * Math.PI)
            {
                toggled = true;
                return simple(ePosRel, pPosRel, vRel);
            }
            lastDir = uRel.clone();
            return uRel;
        }
        else
        {
            var S = Geom.rayCircleIntersection(vRel, ePosRel, Main.R
                );
            var l0 = Math.floor(S.distanceTo(ePosRel) / Main.B);
            var S1 = ePosRel + l0 * vRel;
```

```
            return (S1 - pPosRel).normalizeTo(Main.A);
        }
    }
}
```

EvaderStrats.hx

```
package;

import Demonstration.EvaderStrategy;
import hxmath.math.Vector2;

class EvaderStrats
{
    public static var lastCornerDir:Null<Int>;

    public static function simple(ePosRel:Vector2, pPosRel:Vector2):
       Vector2
    {
        var v = new Vector2(Main.B, 0);
        if ((ePosRel + v).lengthSq <= Main.Rsq)
            return v;
        else
            return onBorder(ePosRel, pPosRel);
    }

    public static function onBorder(ePosRel:Vector2, pPosRel:Vector2
       ):Vector2
    {
        var candidates = Geom.circleIntersections(Vector2.zero,
           ePosRel, Main.R, Main.B);
        var dir = candidates[0].x > candidates[1].x? 0 : 1;
        var destination = candidates[dir];
        return destination - ePosRel;
    }

    public static function onBorderStubborn(ePosRel:Vector2, pPosRel
       :Vector2):Vector2
```

```
{
    var candidates = Geom.circleIntersections(Vector2.zero,
       ePosRel, Main.R, Main.B);
    if (lastCornerDir == null)
        lastCornerDir = candidates[0].x > candidates[1].x? 0 :
           1;
    for (i in 0...2)
        if (candidates[i].distanceTo(pPosRel) <= Main.A)
            lastCornerDir = 1 - i;
    var destination = candidates[lastCornerDir];
    return destination - ePosRel;
}

public static function ratio(ePosRel:Vector2, pPosRel:Vector2)
{
    var dr = ePosRel - pPosRel;
    var expectedCollision = Geom.rayCircleIntersection(dr,
       ePosRel, Main.R);
    var d = (expectedCollision - ePosRel).length;

    if (ePosRel.length > Main.R - Main.B)
        return simple(ePosRel, pPosRel);

    var ro = dr.length;
    var phi = expectedCollision.signedAngleWith(dr);
    var psi = (Math.PI / 2 - phi) * (1 - 1 / (1 + ro / (0.01 *d)
       ));
    var unnormalized = dr.clone().rotate(psi, Vector2.zero);
    return unnormalized.normalizeTo(Main.B);
}

public static function withNoise(ePosRel:Vector2, pPosRel:
   Vector2)
{
    var angleIntervalRadius:Float = Math.PI / 3;
    var v = simple(ePosRel, pPosRel);
    var rnd = Math.random();
```

```
        if (ePosRel.lengthSq < (Main.R - Main.B)*(Main.R - Main.B))
            if (rnd < 0.5)
                return v.clone().rotate(angleIntervalRadius * (Math.
                    sqrt(2 * rnd) - 1), Vector2.zero);
            else
                return v.clone().rotate(angleIntervalRadius * (1 -
                    Math.sqrt(2 * (1 - rnd))), Vector2.zero);
        else
            return v;
    }
}
```

Geom.hx

```
package;

import hxmath.math.Matrix2x2;
import hxmath.math.Vector2;

class Geom
{
    public static function circleIntersections(center1:Vector2,
        center2:Vector2, r1:Float, r2:Float):Array<Vector2>
    {
        var oo1 = center2 - center1;
        var d = oo1.length;
        var cosphi = (r1 * r1 + d * d - r2 * r2) / (2 * r1 * d);
        var sinphi = Math.sqrt(1 - cosphi * cosphi);
        var tildaA = new Vector2(r1 * cosphi, r1 * sinphi);
        var tildaB = new Vector2(r1 * cosphi, -r1 * sinphi);
        var psi = Math.atan2(oo1.y, oo1.x);
        var cospsi = Math.cos(psi);
        var sinpsi = Math.sin(psi);
        var rotationMatrix:Matrix2x2 = new Matrix2x2(cospsi, sinpsi,
            -sinpsi, cospsi);
        var hatA = rotationMatrix * tildaA;
        var hatB = rotationMatrix * tildaB;
```

```
        var A = hatA + center1;
        var B = hatB + center1;
        return [A, B];
    }

    public static function rayCircleIntersection(dir:Vector2, start:
        Vector2, radius:Float)
    {
        var phi = Math.abs(start.signedAngleWith(dir));
        var psi = Math.PI - phi;
        var addend1 = start.length * Math.cos(psi);
        var addend2 = Math.sqrt(addend1 * addend1 - (start.lengthSq
            - radius * radius));
        var x1 = addend1 + addend2;
        var x2 = addend1 - addend2;
        var dirOrth = dir.normalize();
        if (x2 < 0)
            return start + dirOrth * x1;
        else if (x1 < 0)
            return start + dirOrth * x2;
        else if (x2 == 0)
            return start + dirOrth * x1;
        else
            return start + dirOrth * x2;
    }

    public static function randomPointInsideCircle(center:Vector2, r
        :Float):Vector2
    {
        var x = Math.random() * 2 * r + center.x - r;
        var height = Math.sqrt(r * r - x * x);
        var y = Math.random() * 2 * height + center.y - height;
        return Math.random() < 0.5? new Vector2(x, y) : new Vector2(
            y, x);
    }
}
```

MovingCS.hx

```
package;

import hxmath.math.Matrix2x2;
import hxmath.math.Vector2;

class MovingCS
{
    public static var phi:Float;
    private static var sinphi:Float;
    private static var cosphi:Float;

    public static function set(ePos:Vector2, pPos:Vector2)
    {
        phi = (ePos - pPos).angle;
        sinphi = Math.sin(phi);
        cosphi = Math.cos(phi);
    }

    public static function toRelative(v:Vector2):Vector2
    {
        var rotationMatrix:Matrix2x2 = Matrix2x2.fromArray([cosphi,
           -sinphi, sinphi, cosphi]);
        return rotationMatrix * v;
    }

    public static function toAbsolute(v:Vector2):Vector2
    {
        var rotationMatrix:Matrix2x2 = Matrix2x2.fromArray([cosphi,
           sinphi, -sinphi, cosphi]);
        return rotationMatrix * v;
    }
}
```

Drawer.hx

```
package;
```

```
import hxmath.math.Vector2;
import openfl.system.Capabilities;
import openfl.geom.Point;
import openfl.display.DisplayObjectContainer;
import openfl.display.Sprite;

enum PointType
{
    Pursuer;
    Evader;
    Focal;
}

enum TrajType
{
    Pursuer;
    Evader;
}

class Drawer
{
    public static function drawPoint(type:PointType, addOnto:
       DisplayObjectContainer):Sprite
    {
        var color = switch type
        {
            case Pursuer: 0x00FFFF;
            case Evader: 0xFFFF00;
            case Focal: 0x00FF00;
        }

        var point:Sprite = new Sprite();
      point.graphics.beginFill(color);
        point.graphics.drawCircle(0, 0, 4);
        addOnto.addChild(point);
        return point;
    }
```

```
    public static function drawTraj(type:TrajType, start:Vector2, bg
       :Sprite, addOnto:DisplayObjectContainer):Sprite
    {
        var color = switch type
        {
            case Pursuer: 0x0000FF;
            case Evader: 0xFF0000;
        }

        var traj:Sprite = new Sprite();
      traj.x = bg.x;
      traj.y = bg.y;
      traj.graphics.lineStyle(3, color, 0.5);
      traj.graphics.moveTo(start.x, start.y);

        addOnto.addChild(traj);
        return traj;
    }

    public static function drawBG():Sprite
    {
        var bg:Sprite = new Sprite();
      bg.x = Capabilities.screenResolutionX / 2;
      bg.y = Capabilities.screenResolutionY / 2;
      bg.graphics.lineStyle(4, 0x000000);
      bg.graphics.drawCircle(0, 0, Main.R);
      return bg;
    }
}
```