ВЫСШАЯ ШКОЛА МАЙНОР
Институт инфотехнологии
Веб программирование

Александр Мочёнов
IT-3-Q-V-Tal

Искусственный интеллект в играх на примере игры “Войны планет”

Курсовая работа

Руководитель: Zahhar Kirillov, MSc

Таллинн 2010

Оглавление

РЕЗЮМЕ

Данная работа состоит в целом из: 1 частей, более чем 6300 слов, 0 таблицы и 0 картинок.

Ключевые слова: *искусственный интеллект*, *игра*, *теори игр*, *google ai challenge*

В данной работе рассматриваются вопросы и проблемы создания игрового Искусственного Интеллекта. Эта область науки является сравнительно молодой областью, и в ней до сих пор много вопросов без ответов, что делает ей актуальной и привлекательной для автора. Сегодня искусственный интеллект есть во всех играх, и их популярность напрямую зависит от качества искусственного интеллекта. Это делает тему искусственного интеллекта в играх востребованной.

Целью данной работы является знакомство автора с темой и применение полученных знаний при участии в соревновании “Google AI Challenge” и написании собственного искусственного интеллекта в виде бота для игры “Войны Планет”.

В дополнение к основной целе работы автор ставит перед собой следующие задачи:

- Изучение и применение на практике компьютерной верстки LaTeX;
- Переписывание игрового мотора, доработка визуализатора игры и прочее при подготовки окружения для разработки;
- Проверить применимость знание о военном ремесле при разработки стратегии ведения игры;
- Использование системы версионирования исходного кода *git* при разработки программы;

В ходе работы автор приходит к выводу, что для его задачи современные способы написания искусственного интеллекта не подходят. Структура полученного искусственного интеллекта представляет собой стратегический план и делиться на 6 частей: первый шаг, самооборона, защита своих войск, атака, перегруппировка и хитрость. Программный код составляет более 1000 строк, написанных на языке Python.

В ходе испытаний автор убедился в жизнеспособности своего алгоритма, что является главным выводом данной работы.

ВВЕДЕНИЕ

С созданием первого программируемого компьютера сразу после окончания Второй Мировой Войны (1940-1950 гг.) возникла область науки изучающая компьютерные программы, обладающие способностью “мыслить” в том или ином смысле. Эту науку и технологию назвали *Искусственный интеллект* (ИИ). Её основоположниками являются многие учёные, стоящие за созданием тех же первых компьютеров (Алан Тюринг, Джон Маккарти, Марвин Мински и др.)

История ИИ знала более 5 периодов упадка (т.н. “Зима в ИИ”, англ. “`AI winter`”) и столько же возрождений, каждый раз меняя вектор в изучения ИИ. Поэтому до сих пор в этой науке остаётся множество неразрешённых проблем, а добиться результата может каждый (Russell and Norvig, 1995), в отличии от более старых и фундаментальных областей вроде физики и химии. “Искусственный интеллект, с другой стороны, всё ещё открывает возможности для проявления талантов нескольких настоящих Эйнштэйнов” (Russell and Norvig, 1995) Это всё делает ИИ для автора исключительно интересной областью для изучения.

Практически сразу после основания ИИ появились первые программы, играющие в игры. В 1957-ом году Кристофер Стрэчи написал игру, играющую в шашки. Тогда же был написан первый ИИ, примитивно играющий в шахматы. (Russell and Norvig, 1995) ИИ в компьютерных играх с тех пор значительно развился. Тем не менее, шахматную программу сильнее человека смогли написать лишь в 1997-ом году, когда DeepBlue одержала победу над Гарри Каспаровым. Это стало возможным лишь благодаря увеличению компьютерных мощностей. Но этот успех был возможен также благодаря новым алгоритмам и подходам[1]. В любом случае не стоит полагать, что развитие компьютерной техники является “серебряной пулей” в решении всех проблем в компьютерных ИИ. Например, игра “Го”[2] до сих пор является нерешённой задачей для программистов ИИ. Самая умная на сегодняшний день играет в неё на любительском уровне. (Russell and Norvig, 1995)

Сегодня индустрия компьютерных игр практически не обходится без применения ИИ. Он в играх, на ряду с графическими и прочими характеристиками, является важным качественным показателем и напрямую влияет на их конкурентно-способность и успех. Это делает игровой ИИ востребованным. Игры по своей природе зачастую имеют не

[1] Хотя сами создатели не считали свой творение “интеллектуальным”Moravec (1998)

[2] Го - это стратегическая настольная игра. Го является одной из наиболее распространённых настольных игр на Земле GoGameWiki

только развивающую, но и развлекательную составляющую, по-этому автору данная тема особенно интересна.

Целью автора является знакомство с областью науки ИИ, изучение её основ с последующим применением полученных знаний на практике в ходе разработки и написания собственного ИИ, играющего в игру с другими себе подобными. Для практического задания автор участвует в соревновании “Google AI Challenge” (`http://ai-contest.com/`), в ходе которого разрабатывает ИИ для игры “Войны планет”. Написание программы, которая играет в игру в качество одно из игроков, обладающему знаниями в программировании автору, представляется не только интересным, но и весёлым занятием. Такую программу принято называть *ботом* (сокращение от слова робот). Методология, применяемая автором, в основном квалитативна и выводы по большей части представляют из себя, основанное на полученных результатах, субъективное мнение автора.

ИИ является чрезвычайно широкой сферой науки, и для написания игрового ИИ можно применять множество кардинально отличных друг от друга подходов и алгоритмов. Для написания ИИ автор использует *символьно-логический*, *агенто-ориентированный* подход без алгоритмов поиска. Автор не делает глубокого анализа, принимая это решение, что бы не выйти за рамки допустимых объёмов работы. Тем не менее он объясняет причину своего выбора. Так же в работе применяются понятия из *теории игр*.

“Войны планет” – так называется игра для которой автор пишет ИИ. Как следует из названия в каком-то смысле игра о войне. По-этому как дополнение к написанию своего бота, автор ставит задачу проверки применимости знаний о военном ремесле, взятых из древнего трактата Сунь Дзы “Искусство войны”, к дизайну стратегий и тактик ИИ.

Так же автор ставит задачу изучения системы компьютерной вёрстки LaTeX, применения её в ходе написания данной работы. Эта система широко применяется в научной среде для написания статей, книг и научных работ, по-этому такие знания должны быть полезны автору в целом и как тренировка перед написанием дипломной работы.

Для эффективного управления версиями исходного кода при написании используется система версионирования git. А для вспомогательных программ (всё за исключением исходного кода самого бота) автор выложил на сайте GitHub[3] (`http://github.com/soswow/python-ai-challenge`).

Данная работа будет полезна всем, кому интересна тема ИИ. Ознакомившись с данной работой, можно сложить представление о том, что можно сделать и с чего можно начать, если стоит задача в написании игрового ИИ.

[3]Сайт на котором можно делать git хранилище и делиться открытым исходным годом с обществом

1 ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

Прежде чем подойти к описанию выбранной игры и непосредственному написанию игрового ИИ для неё, необходимо ознакомиться с некоторыми теоретическими аспектами вопроса. Так что же такое ИИ? Чёткого ответа на этот вопрос нет. Есть 4 основных вида определений, которые можно обобщить следующим образом: (Russell and Norvig, 1995) Искусственный интеллект это:

- Системы, которые думают подобно людям
- Системы, которые думают рационально
- Системы, которые действуют подобно людям
- Системы, которые действуют рационально

Есть мнение, что искусственный интеллект даёт возможность компьютерам выполнять мыслительные задачи, на которые способны люди и животные.(Millington and Funge, 2009)

“Может ли машина мыслить?” – этим вопросом задался Алан Тюринг в 1950 году в статье “Вычислительная машина и интеллект” (Alan, 1950), в которой он предлагает тест на способность машины проявлять интеллектуальность. Суть теста такова: человеку даётся возможность пообщаться с кем-то через текстовую консоль (чат), после чего человека спрашивают - говорил ли он с ИИ или с человеком? Если ИИ заставляет поверить, что он - настоящий человек или даже просто засомневаться, то тест считается пройденным. Этот тест является основой в *философии об искусственном интеллекте*, хотя и подвергался неоднократной критике. (Searle, 1980) На данный момент ни один ИИ не справился с ним на 100 процентов.

1.1 Подходы к написанию ИИ

Как уже было сказано выше ИИ развивается с 50-ых годов XX века. За это время сложилось два основных направления в написании ИИ.

1.1.1 Символьный и логический подходы

Символьный подход в написании ИИ сформировался в начале 50-ых. Этот подход заключается в разделении ИИ на две части: *база знаний* и набор механизмов, которые этими знаниями манипулируют, что бы получить решение проблемы или новые знания. Такими механизмами часто являются *алгоритмы поиска*.

ИИ такого типа может обладать большой базой знаний и не очень сложными алгоритмами. Таковыми являются, например, *экспертные системы* (ЭС). ЭС - это программы, которые заменяют человека-эксперта в какой-либо области. Их цель - дать заключение (решение) для задачи, исходя из полученных входных данных, используя имеющуюся базу знаний, которая пополняется или вручную или автоматически, если система обладает способностью к самообучению.

ИИ можно сделать также с небольшим количеством знаний (или их полным отсутствием) и мощными алгоритмами поиска. Такие алгоритмы не полагаются на уже имеющиеся знания, а пытаются найти решение проблемы обладая только входными данными. Такой подход часто становиться основой для ИИ в настольных играх (шашки, реверси, шахматы).

Возможны и комбинированные решения. Так, например, вышеупомянутая программа DeepBlue умела как очень эффективно искать[1], так и обладала базой знаний эндшпилей, дебютов. А также применяла базу данных из 700 000 игр гроссмейстеров, из которых программа брала согласованные рекомендации. (Russell and Norvig, 1995)

1.1.2 Квазибиологическая парадигма

До 80-ых годов символьный подход служил опорой для написания ИИ, которые решали не сложные задачи, что с учётом компьютерной техники того времени считалось большим достижением. Когда набор проблем, которые пытались решить с помощью ИИ начал расширяться и усложняться, символьно-логический подход уже не мог справиться с многими из них.

Символьному на смену пришла эпоха моделирования биологических систем, как с применением реальных биологических элементов (Рамбиди et al., 2002), так и просто копи-

[1] до 30 миллиардов позиций в расчёте на каждый ход, обычно достигая глубины поиска 14

рованием систем и процессов из природы. К таким алгоритмам в частности относятся:

- Искусственные нейронные сети
- Генетические алгоритмы
- Эволюционные вычисления

Искусственные нейронные сети (ИНС) представляют собой попытку скопировать взаимосвязи и принципы действия биологических *нейронов* (нервных клеток). ИНС представляют собой математическую функцию, где на вход подаётся некий вектор; ИНС пропускает данные через ряд межнейронных связей, преобразуя данные и на выход выдаёт некий выходной вектор, что и является решением задачи.Haykin (1994)

Генетические алгоритмы и эволюционные вычисления копируют или напоминают эволюционные механизмы в природе. Применяются такие понятия как: мутации, скрещивание, индивидуум, популяции и прочее. В процессе работы алгоритма на разных этапах поиска лучшие решения могут совмещаться, образую новое, улучшенное поколение, а какие-то решения могут случайных образом меняться, т.е. мутировать, внося тем самым положительный стохастический элемент.(Russell and Norvig, 1995)

Стоит заметить, что нейронные сети, например, были предложены и разработаны ещё в 1943 г. (McCulloch and Pitts, 1943) Кто-то считает движение от символьного исчисления к биологическим прогрессом, а кто-то полагает, что это просто дань моде, разной в разные времена. (Millington and Funge, 2009) Автор лично придерживается мнения, что какие-то задачи хороши для решения старым, проверенным методом символьной логики. При этом с использованием биологических моделей решение получается не столь эффективным и требует больше усилий, а другие наоборот. Другие же задачи наоборот лучше решать современным методами. Например многие задачи компьютерного зрения хорошо решаются искусственными нейронными сетями. (Haykin, 1994)

1.1.3 “Хороший” или развлекательный игровой ИИ

Все ИИ написанные для игр можно разделить на две категории: “Хороший” и развлекательный ИИ.(Johnson, 2010) При этом “хороший” тут не качественный показатель, а имя собственное. Это зависит от типа игры и от ожиданий игрока.

Основной задачей “хорошего” ИИ является победить игрока-соперника, будь-то человек или другой ИИ. Примером такого ИИ может служить программа играющая в шахматы, шашки и другие настольные игры. Люди играют с таким ИИ с целью или

попрактиковаться или они просто испытывают напряжение играя с людьми. (Johnson, 2010)

Целью же развлекательного ИИ, как следует из названия, является развлечь игрока. Таким ИИ обычно обладает большинство современных игр. В таких играх как “Sims” или “Half-Life” есть компьютерные игроки, которых называют “персонажами”. Задача программистов этих игр сделать персонажей максимально реалистичными и ведущими себя как если бы это был реальный герой. Например, если по персонаж - охранник, то он не должен забывать про игрока, как только он пропадёт из поля зрения игрока, а должен стараться подать сигнал тревоги. Игрок не должен ощущать, что с ним играет компьютер, который пользуется информацией недоступной самому игроку. В противном случае интерес к игре пропадёт. Что бы обеспечить адекватность и рациональность решений и действий персонажей с ИИ, программисты, например, ограничивают доступ к информации об окружающем мире: ИИ может “видеть” только то, что находиться в поле его зрения; ИИ может слышать звуки, что слышат все. Т.е. объём информации доступной ИИ должен быть схож с тем, что может позволить себе игрок. (Buckland, 2005)

1.2 Агентно-ориентированный подход

Агентно-ориентированный подход (АОП) в написании ИИ появился в 1990 г. Он со состоит в разработке *агентов*, которые составляют суть всего АОП. “Агентом является всё, что может рассматриваться как воспринимающее свою *среду* с помощью *датчиков* и воздействующее на эту среду с помощью *исполнительных механизмов*” (Russell and Norvig, 1995) Т.е. ИИ должен иметь доступ к информации о среде в которой он функционирует, на которую он сможет влиять каким-то способом. Схематично это показано на рис 1.1.

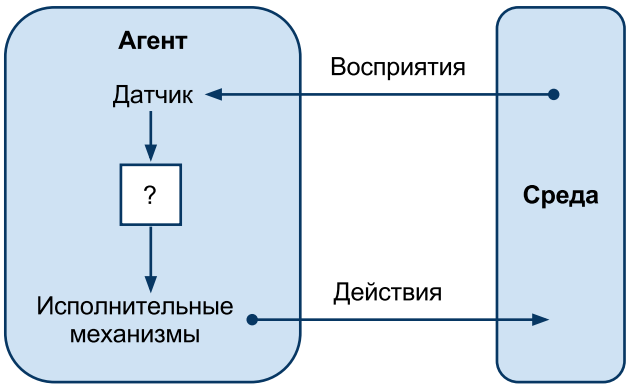


Рис. 1.1: Агент взаимодействует со средой с помощью датчиков и исполнительных механизмов (Russell and Norvig, 1995)

Такой ИИ представляет собой математическую *функцию агента*, реализовать которую должен программист в виде *программы агента*. Агент может накапливать информацию о полученных состояниях среды, что называется *последовательностью актов восприятия*. Он может использовать их при принятие решения и осуществлении ответных действий.

Для успешного выполнения поставленной задачи агент должен быть *рациональным*. “Рациональность - это максимизация ожидаемой производительности, а совершенство - максимизацией фактической производительности” (Russell and Norvig, 1995) Рациональный выбор агента зависит только от последовательности актов восприятия, сформированного к данному моменту. Например, если человек переходит дорогу и не посмотрит по сторонам (и попадёт под машину) - это будет нерациональный поступок, т.к. он мог это сделать и повысить ожидаемую производительность. А если он посмотрим по сторонам, и при этом на него упадёт метеорит, то это будет всё равно рациональный поступок.(Russell and Norvig, 1995)

Что бы говорить о рациональности, как следует из определения, необходимо определить *критерии производительности*. Для каждого агента этот показатель будет зависеть в основном от среды и от задачи, которую он будет решать. По сути производительность - это на сколько хорошо агент справляется с поставленной задачей. По-этому, что бы написать программу агента надо чётко представлять себе среду в которой он функционирует, механизмы восприятия и воздействия на ней.

Как будет видно из 2 части, механизмами восприятия и воздействия в данной игре будет являться программный API. Описание конкретной среды, в которой находиться ИИ и критерии производительности так же будут описаны ниже.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

Для того чтобы более конкретно представить себе хотя бы частично как создаётся ИИ и с какими проблемами приходиться сталкивать автор выбирает в качестве практического задания написание ИИ для какой-либо игры. Для этого автор принимает участие в соревновании “Google AI Challenge” (`http://ai-contest.com/`), организованном компьютерным клубом при Университете Уотерлоу (англ. “`University of Waterloo`”). Компания Google является лишь спонсором проводимого мероприятия. (AIChallangeFAQ)

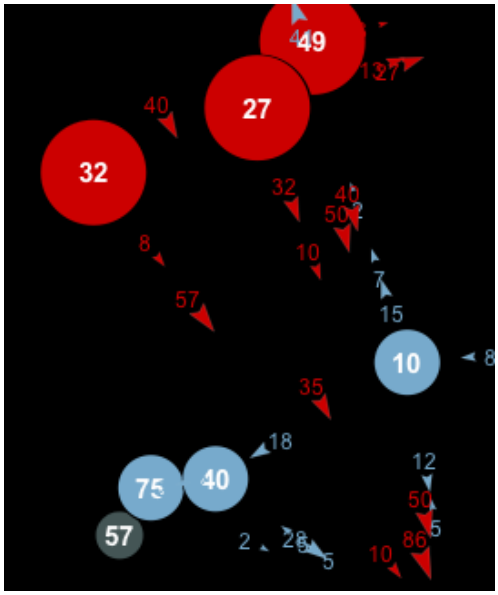
Данное соревнование является одним из многих, суть которых заключается в написании ИИ для игры с двумя или более игроками. Участник конкурса должен написать по сути программу агента, которая будет реализовать функцию агента, что часто необходимо сделать в буквальном смысле, реализовав какой-то итерфейс с одной функцией вроде “`Decition doTurn(gameState)`”. После чего ИИ сражаются друг с другом.

Подобные конкурсы встречаются и у нас в Эстонии. Например, на первом курсе Таллинского Технологического Университета на предмете по программированию в конце курса всем студентам предлагается написать свой ИИ для игры в шашки, после чего все написанные ИИ сражаются друг с другом в турнире.

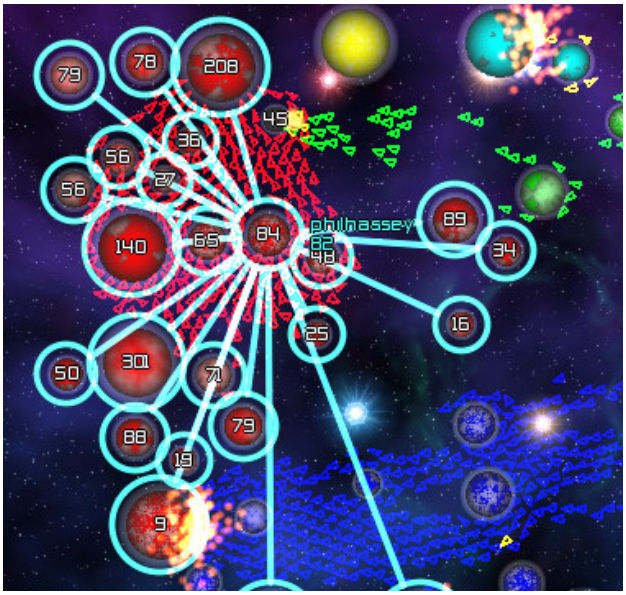
Подобные соревнования представляются автору наиболее интересным видом практического задания по нескольким причинам. Во-первых это возможность применить знания о программировании и об ИИ. Во-вторых здесь присутствует соревновательский дух, что является несомненно положительным вектором. И на конец речь идёт об играх - а значит это интересно и весело.

2.1 Об игре “Войны планет“

Игра “Войны планет” (далее ВП или просто *игра*) - это пошаговая стратегическая игра для 2-ух и более игроков (рис. 2.1a). Она основана компьютерной игре “Galcon” (`http://en.wikipedia.org/wiki/Galcon`, рис. 2.1b), специально переработанная для соревнования. В частности, в оригинале игра реального времени, что для упрощения процесса было заменено пошаговым стилем, давая тем самым право хода каждому игроку-боту по очереди.



(a) Визуализатор игры на сайте конкурса



(b) Flash игра “Galcon” на официальном сайте

Рис. 2.1: Фрагменты игр “Войны планет” (a) и “Galcon” (b)

2.1.1 Сущности игры

ВП представляют собой 2D холст или карту, где происходит сама игра, на котором действуют следующие сущности (рис. 2.2):

Планета

Планета может

- принадлежать одному из игроков или быть нейтральной;
- хранить, и если принадлежит кому-то, то и производить корабли с константной скоростью[1];
- выпускать корабли для атаки противника, захвата нейтральных планет или для перегруппировки;
- становиться чьей-то в случае нейтральности или менять хозяина в случае успешной атаки;

В начале игры все планеты располагаются на карте и до конца игры не меняют своего положения. *Расстоянием* между планетами считается как округлённое геометрическое расстояние, рассчитанное по координатам планет. Планеты располагаются таким образом, что как бы делят карту на две симметричные половины. Все планеты кроме двух изначально нейтральные и обладают определённым

[1] *Коэффициент роста* или просто *рост* - это то, сколько кораблей будет появляться на планете за один ход. Обычно варьируется от 0 до 5.

количеством кораблей. У каждого участника в начале игры есть в наличии по одной планете с сотней кораблей.

Флотилия и корабль

Флотилия - это группа кораблей, которые *планета-источник* пустила по направлению к *планете-цели*. Флотилия существует количество ходов, равное расстоянию между планетами источника и цели, перемещаясь за каждый ход на одну единицу[2] по направлению к цели. С момента выпуска флотилии с ней ничего нельзя сделать: ни поменять направления, ни отменить решения. *Корабль* - это разменная единица, из которых “состоят” флотилии и которые производят планеты участников.

Выбор

Решение, которое принимает ИИ игрока на каждом *полу-ходу*[3] партии (см. 2.3). Он состоит из набора новых флотилий, выпускаемых на этом ходу игроком.

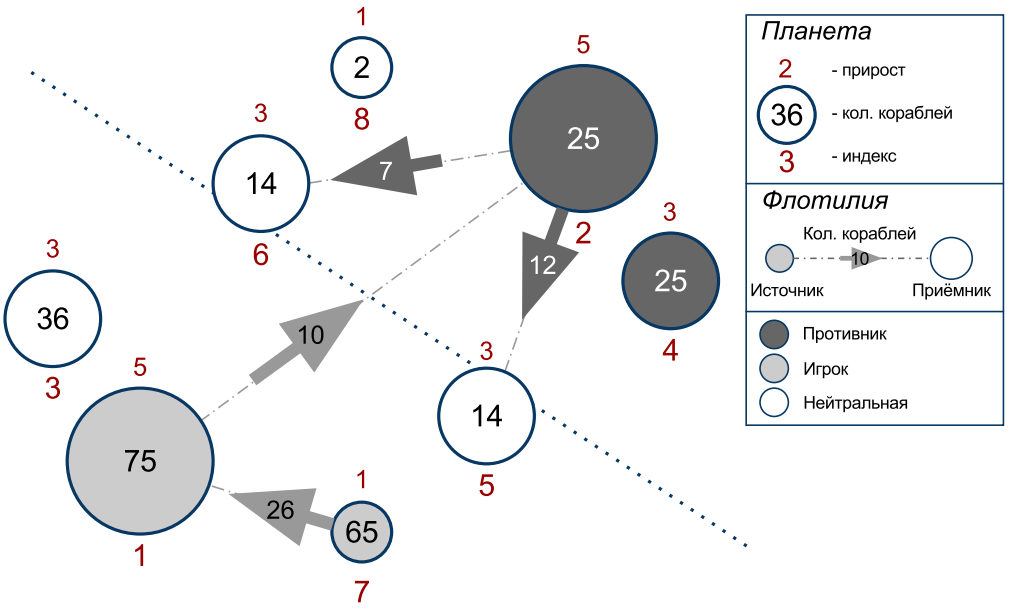


Рис. 2.2: Схематичное представление сущностей игры. Легенда справа.

На рисунке 2.2 схематично изображена карта игры со всеми её сущностями. Планеты 2 и 4 принадлежат одному игроку, а 1 и 7 другому. Остальные планеты (3,5,6,8) - нейтральные. Точечной диагональной линией показана воображаемая линия симметрии. Стрелочками обозначены выпущенные или уже летящие флотилии кораблей. В частности, игрок, играющий светлыми, на этом ходу выпускает флотилию из 26-и кораблей с планеты 7 по направлению планеты 1 (7-1). А флотилия выпущенная ранее вторым

[2]Имеется ввиду единица расстояния. Например, если расстояние между планетами равняется 5, то флотилия пролетает от одной до другой за 5 ходов.

[3]Это половинка всего хода, состоящая из решения одного из игроков

игроком со 2-ой планеты на планету 6 перемещается на одну единицу. Таким образом выбором первого игрока является отсылка одной флотилии 7-1, а выбором второго флотилия 2-5. В визуализаторах размер планеты пропорционален зависит от её *росту*.

2.1.2 Правила игры

Игра разделена на ходы. Каждый ход состоит из двух полу-ходов для каждого из игроков. Во время полу-хода игрок делает выбор, отдавая *приказания* о выпуске новых флотилий, указывая планеты отправителя, планету назначения и количество кораблей в каждой флотилии. Планетами отправителя могут быть только свои планеты, а количество кораблей в отправляемой флотилии не должно превышать количество имеющихся на планете. В момент отправки флотилии количество кораблей на планете уменьшается уменьшается на размер флотилии.

Когда флотилия достигает места назначения, она исчезает, изменяя статус для планеты-цели. Это изменение зависит от взаиморенадлежности флотилии и планеты-цели к игрокам. Могут произойти следующие варианты:

- Если флотилия прилетела на свою же планету, то корабли флотилии добавляются к кораблям на планете.
- Если флотилия прилетела на планету противника, то корабли взаимно уничтожаются и на планете остаётся разница от вычитания кораблей во флотилии и на планеты. В случае, если кораблей больше во флотилии, то планета меняет хозяина на того чья флотилия это была.
- Если на планету в один ход прилетает сразу несколько флотилий разных принадлежностей, то работают более сложные правила, которые можно найти на сайте соревнования (PlanetWarsSpec).

Цель игры - захватить как можно больше планет за максимум 200 ходов. На каждый ход игроку даётся одна секунда. Игра заканчивается или по истечению одного из лимитов, или если у одного из игроков закончатся корабли (как на планетах, так и во флотилиях). (PlanetWarsSpec)

2.1.3 Технические данные

Для участия в соревновании необходимо зарегистрироваться на сайте соревнования. Далее надо скачать начальный пакет, состоящий из игрового *движка*[4] (написанного на

[4]Так автор называет программу, отвечающую за коммуникацию между ботами, хранение и обновление состояния игры. (англ. “`Engine`”)

Java), нескольких очень простых ботов, набора из сотни карт и шаблона для написания своего бота. Для разных языков программирования готовых стартовых пакетов имеется четыре варианта: для C++, Java, Python, C#. Основной движок конкурса поддерживает запуск ботов, написанных также на: Haskell, Ruby, Javascript, PHP, Perl, OCaml и Lisp'е. Стартовые пакеты для них можно найти на форуме соревнования. Автор другим языкам предпочитает Python, по-этому именно на нём он пишет своего бота.

Для того, что бы участвовать в конкурсе в одном из файлов необходимо заполнить логикой метод "`def DoTurn(pw): ...` ", где на вход поступает объект в котором имеется API для доступа ко всей информации об игровом состоянии среды. С помощью этого API можно узнать где, какие планеты располагаются, сколько у кого кораблей, от куда куда движутся флотилии, когда прибудут и т.д.

После реализации бота все необходимые файлы для его запуска надо запаковать в архив и залить на официальный сервер, где бот игрока будет периодически играть с другими ботами, имеющих примерно тот же рейтинг. В зависимости от исхода каждой партии меняется количество очков участника, что сдвигает его в общей рейтинговой таблице.

Во время игры обмен информацией о состоянии игры и выборах игроков в игровом движке происходит следующим образом (рис. 2.3):

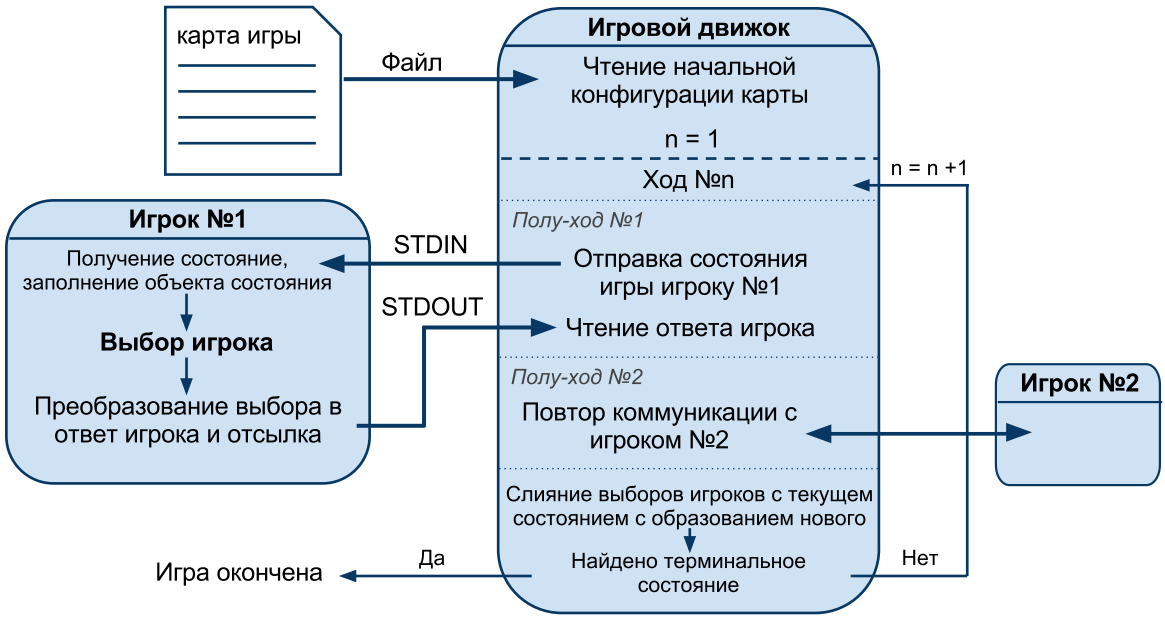


Рис. 2.3: Процесс работы игрового движка

На рисунке 2.3 видно, что в начале игры читается файл с описанием начальной карты, которая составляет начальное состояние игры. Это состояние передаётся первому игроку, который делает свой выбор и отдаёт назад приказы об отправлении новых флотилий, после чего всё тоже самое повторяется для второго игрока. Далее движок

обрабатывает полученные *выборы*, применяя их к текущему состоянию игры, а так же он делает передвижение уже имеющихся флотилий, происходит прирост кораблей на планетах и обработка прибытий флотилий на планеты. После чего он проверяет терминальное состояние, т.е. не закончена ли игра (см. раздел 2.1.2).

Перед началом игры движок запускает программы игроков в отдельных процессах и устанавливает с ними связь через *стандарт ввода-вывода* (англ. "`Standart Input-Output, STDIN/OUT`"), в последствии общаясь по ним с процессами ботов, передовая состояния и зачитывая выборы игроков.

2.2 Подготовка к написанию ИИ

Прежде чем приступить к написанию бота, необходимо решить некоторые дополнительные задачи: подготовить дополнительные средства для помощи в разработки; определиться со средой, в которой функционирует бот для конкретной игры; выбрать методологию написания ИИ и методы принятия решения о выборе;

2.2.1 Подготовка окружения для разработки

В ходе первых попыток написания собственного бота, автор столкнулся с рядом сложностей, в связи с чем составил список того, что необходимо сделать для успешного завершения проекта прежде чем приступать к непосредственному писанию ИИ.

Журналирование

Система при которой игровой движок общается с ботами в виде отдельных процессов через стандартный ввод-вывод означает отсутствие возможности *пошаговой отладки* (англ. "`debug`") даже во время тестирования на локальном компьютере. По-этому автор с самого начала создал систему *журналирования* (или логов, от англ. `logs`), которая достаточно полно раскрывала процессы, происходящие во время игры.

Свой движок

Чтение логов утомительный процесс и занимает слишком много времени. Для решения этой проблемы и для более глубокого понимания процессов, происходящих в движке игры автор решил переписать движок на языке Python. Это позволило подключать ИИ напрямую к движку в качестве библиотеки, тем самым давая возможность для отладки.

Модификации в визуализаторе

Так же для удобства разработки автор модифицировал визуализатор сыгранных игр, написаны на Java и поставляемый в стандартном наборе (рис. 2.4).

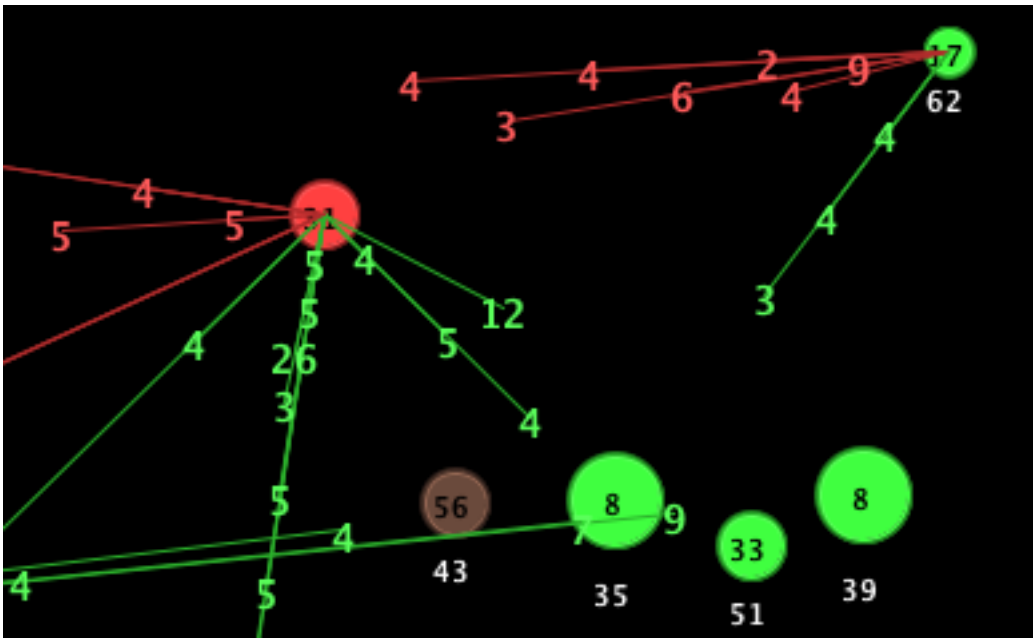


Рис. 2.4: Фрагмент модифицированного визуализатора

В частности:

- Была добавлена возможность менять скорость воспроизведения при помощи дополнительного аргумента во время вызова приложения. Изначально она была слишком медленной.
- Были добавлены линии, показывающие траекторию полёта флотилий. Часто было не понятно какая цифра куда движется.
- Была добавлена возможность передавать визуализатору дополнительную отладочную информацию касательно планет с последующим её выводом на экран (на рисунке белый цифры под планетами). Например при выборе цели для каждой планеты считается коэффициент, найти и сопоставить с конкретной планетой из логов было достаточно сложно.

Написание тестов

В подобной задаче безусловным подспорьем являются *модульные тесты* (англ. “`unit tests`”), которые также присутствуют в написанном ИИ

Система игры на всех картах

Для эффективной проверки производительности разрабатываемого ИИ был написан несложный *сценарий* (англ. “`script`”) для запуска бота на всех картах из начального набора. В конце работы он выдаёт статистическую информацию по тому сколько игр было выиграно, проиграно и за сколько ходов.

2.2.2 Процесс разработки

Процесс разработки бота можно разделить на несколько этапов:

1. Написание первого бота и тестирование его на всех картах против всех простых ботов, что идут в начальном наборе.
2. Улучшение бота до тех пора, пока он не будет побеждать в 100% случаях.
3. “Замораживание” бота в его текущем состоянии и установка его на место противника и переход к пункту 2. Т.е. теперь задача играть против своего же бота, и улучшая следующее поколение прийти к полной победе над ним.
4. Периодически пользоваться TCP Server’ом (`http://www.benzedrine.cx/planetwars/`). Это неофициальный централизованный сервер, на котором можно в реальном времени сразиться с другими клиентами сервера. Главное отличие от официально сервера заключается в следующем:
   (a) В скорости. Подключившись игра начинается практически сразу, в отличии от официального сервера, где каждый бот играет в среднем раз в пол часа.
   (b) Не нужно заливать свой программный код на сервер. Код бота запускается на компьютере его владельца, тем самым давая возможность записывать логи для последующего анализа.
   (c) Игроки на TCP Server’е зачастую очень сильные, по-этому для бота это всё равно что *стресс тестирование*.
5. Если появляется ощущение, что бот уже достаточно работоспособный, его можно упаковывать и засылать на официальный сервер и переходить к пункту 3.

2.2.3 Выбор типа ИИ и метода его написания

На рисунке 1.1 (стр. 8) показана модель простейшего агента. Если сравнить её с рисунком 2.3 (стр. 14), то можно заметить, что схема игрока №1 имеет те же составные части (вход, выход, принятие решения), что и агент. По-этому автор считает уместным в данной задаче применение агентно-ореинтированного подхода. По сути надо реализовать белый ящик со знаком вопроса на рисунке 1.1 или более формально функцию агента.

Характеристики игры очень похожи на свойства среды, в которых функционирует агент, играющий в шахматы за одним важным исключением. Средний *коэффициент ветвления* в шахматах примерно равен 35. Т.е. в среднем на каждом полу-ходу игрок может выбирать среди 35 возможных решений, что даёт приблизительно 35^{100} различных комбинаций состояний игры. (Russell and Norvig, 1995) При этом программа DeepBlue могла просмотреть только на 14 ходов вперёд.

Изучая вопрос эффективности различных методов и подходов для написания ИИ, автор ознакомился с некоторыми мнениями специалистов. Так, Антон Сафонов (MSc, магистерская работа которого была посвящена комбинированию методов оптимизации роя частиц и монте-карло метода) считает, что данная задача должна решаться “современными способами, т.е. необходимо сделать так, что бы не нужно было говорить программе как решать задачу, а что бы программа сама находила пути её решения”. Т.е. его идея заключалась в применении оптимизированных алгоритмов поиска, абстрагируясь от конкретных планет и флотилий до понятий вроде “скопление сил”, решая тем самым проблему чрезмерного коэффициента ветвления в ВП. Автору такой подход, ввиду отсутствия необходимых познаний в данной области, показался слишком сложным и непонятным. Более того, ему кажется, что именно “ручной” способ написания ИИ в данном случае будет более эффективным.

Для написания агента с использованием конкретных правил и законов, а иначе говоря используя стратегию и тактику, необходимо использовать знания о том “как работает мир”. Такой агент называется *агентов основанным на модели*. (Russell and Norvig, 1995) С учётом того, что автор не собирается использовать алгоритмов поиска, для просмотра всех вариантов развития событий, то для такого агента достаточно разработать систему рефлексов, т.е. механизмы реагирующие исключительно на текущее акты восприятия. Хотя некоторые аспекты алгоритма всё же можно рассматривать как предсказание, когда система защиты строится на предположении относительно некоторых планет (см. 2.3.2).

2.2.4 Характер среды

“Критерии успеха, наряду с описанием среды, а также датчиков и исполнительных механизмов агента, предоставляют полную спецификацию задачи, с которой сталкивается агент” (Russell and Norvig, 1995) Критерием успеха или показателем производительности в случае игры ВП является счёт на конец соревнования в итоговой рейтинговой таблице, а так же результат каждой сыгранной партии. Во время локального тестирования на всех картах такими показателями являются также количество ходов, за которое бот победил и проиграл.

Существуют характеристики, по которым можно классифицировать среды в которых оперируют ИИ. (Russell and Norvig, 1995) В данном случае имеет место форма взаимодействия ИИ-ов, представляющая собой детерминированную, поочередную, охватывающие двух игроков игру с *нулевой суммой* и с *полной информацией*. (Russell and Norvig, 1995; Morgenstern and Von Neumann, 1947) Эти и другие свойства среды и их значения, применительно к данной игре приведены в таблице 2.1 (стр. 19).

Свойство среды	Значение для среды	Почему?
Наблюдаемая полностью или частично	Полностью наблюдаемая	ИИ имеет доступ ко всем данным, необходимых для принятия решения.
Детерминированная, стохастическая или стратегическая	Стратегическая	Является таковой, если “среда является детерминированной во всех отношениях, кроме действий других агентов”(Russell and Norvig, 1995)
Эпизодическая или последовательная	Последовательная	Т.к. выбор игрока на каждом ходу влияет на все будущие решения
Статическая или динамическая	Полудинамическая	Т.к. “с течением времени сама среда не изменяется, а изменяются показатели производительности”. (Russell and Norvig, 1995) Выбор принимается в условиях ограниченного времени.
Дискретная или непрерывная	Дискретная	Т.к. игра “имеет конечное количество различимых состояний” и она “связана с дискретным множеством восприятий и действий“ (Russell and Norvig, 1995)
Одноагентная или мультиагентная	Конкурентная мультиагентная среда	Т.к. имеет место соперничающая сущность Б, которая пытается максимизировать показатели своей производительности за счёт минимизации показателей сущности А

Таблица 2.1: Характеристики среды игры ВП

2.3 Стратегия и тактика

В этом разделе описаны стратегии и тактики, которые автор применил и реализовал в своём агенте. Так же тут делается попытка связать их со знаниями о военном ремесле, описанными в трактате “Искусство Войны” (5-6 вв. до н. э.).

> “Если используешь их[войска] в битве, но победа долго не приходит, их оружие притупляется, а рвение - ослабевает. Если осаждаешь города, их силы истощаются. Если подвергаешь войско длительной войне, запасов государства не хватит ... Поэтому я слышал об успехе быстрых военных походов, и не слышал об успехе затяжных.” (Tzu, 6 век до Н.Э.)

Это правило действительно и тут, потому что игра продолжается максимум 200 ходов. Что бы достичь победы, медлить нельзя. В этой игре быстрота развёртывания сил и захвата планет очень важен. В момент главного сражения разница в одну планету может решить исход исход партии.

Как уже говорилось ранее в начале игры у обоих игроков условия совершенно идентичные. У каждого есть по планете с сотней кораблей, а все остальные планеты находятся на равных расстояниях от начальных планет игроков, т.к. карта симметричная. Из этого можно ещё раз подчеркнуть мысль, что исход каждого поединка зависит исключительно от алгоритмов игроков.

Общую стратегию, можно охарактеризовать как “играть, что бы не проиграть”, т.к. по мнению автора главное в этой игре - правильная оборона, что будет раскрыто ниже (см. 2.3.2).

Вся стратегия автора делиться на 6 этапов или элементов. Алгоритм проходит через все эти этапы принимает решение о выборе. Каждый из них описан далее более подробно.

2.3.1 Первый ход и расширение

Самым первым этапом игры является фаза “развёртывания”, в которой особенно важно самый первый *выбор*. Причина тому в симметричности первоначального состояния игры для обоих игроков. По-этому очень важно с первых секунд игры попытаться получить преимущество, выбрав планеты для первой атаки наиболее эффективно.

> “Тому, кто первым приходит на поле сражения и ожидает врага, будет легко;
> тот, кто приходит после и должен спешить в бой, будет утомлен”
> (Tzu, 6 век до Н.Э.)

В отличае от шахмат в этой игре нету игрока идущего первым, оба игрока делаю свой первый полу-ход не зная о решении противника. Неправильное решение на первом шаге может закончиться быстрой победой противника. Пример на рисунке 2.5.

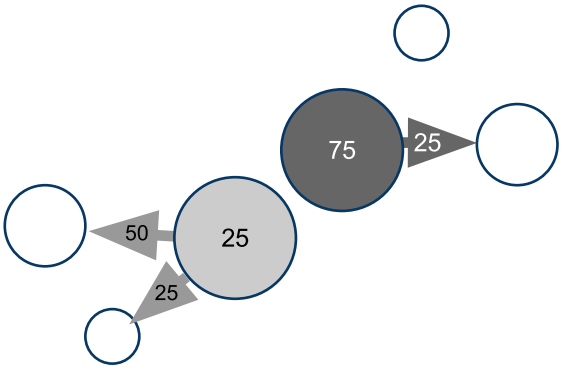


Рис. 2.5: Ошибка первого игрока (светло-серый) на первом шаге

В показанной ситуации начальные планеты игроков находятся очень близко друг от друга и один из участников не обращая на это внимание рассылает бóльшую часть

своих кораблей на захват нейтральных. В то же время его противник не спешит распускать все корабли. В результате ошибки второй игрок следующим ходом захватит изначальную планету первого игрока и получи тем самым победное преимущество, т.к. начальные планеты имеют высокий коэффициент роста (обычно 5).

На самом деле при хорошей самообороне и правильно выбранной *функции выбора цели* для атаки (см. 2.3.4) никакой особой логики в первый ход вкладывать не обязательно (будет видно далее). Хотя автор всё же разработал рефлекс[5] который срабатывает на первом и последующих ходах, если ситуация такая как описана выше. Он оставляет все корабли на первом ходу, а на втором посылает все корабли на главную планету противника.

2.3.2 Самооборона

Самооборону можно считать пожалуй самым важным этапом всей стратегии. От того как хорошо и безошибочно реализован этот модуль алгоритма зависит результативность всего бота. Побеждает тот, у кото суммарный прирост кораблей на всех планетах больше. По-этому потеря любой своей планеты ведёт к изменению этого отношения, а значит этого нельзя допустить ни в кое случае.

> “Поэтому тот, кто преуспел в войне, первым делом выбирает позицию, где он не может быть разбит, вместе с тем не упуская [любой возможности] разбить врага.”
> (Tzu, 6 век до Н.Э.)

Суть самообороны в том, что бы на планете всегда оставалось достаточно кораблей, что бы смочь отбить любую атаку. Для этого прежде всего надо посмотреть сколько и какие флотилии летят на данный момент на планету и представив, что более никакая планета не выпустит флотилий по данному направлению, просиммировать все прилёты кораблей получив в результате чёткое представление о судьбе этой планеты.

Для удобства была написана специальная функция, которая занимается симуляцией таких случаев. В результате она выдаёт информацию о том на каких ходах и как будет меняться состояние этой планеты (количество кораблей, принадлежность) по мере прилёта флотилий. Из этой информации можно сделать определённый и важный вывод - сколько кораблей необходимо забронировать на самооборону, что бы когда прилетит последняя флотилия планета осталась у игрока.

На рисунке 2.6 упрощенно объясняется как работает алгоритм симуляции.

[5] и назвал его “блицкриг”, хотя название “во-банк” тут тоже подходит

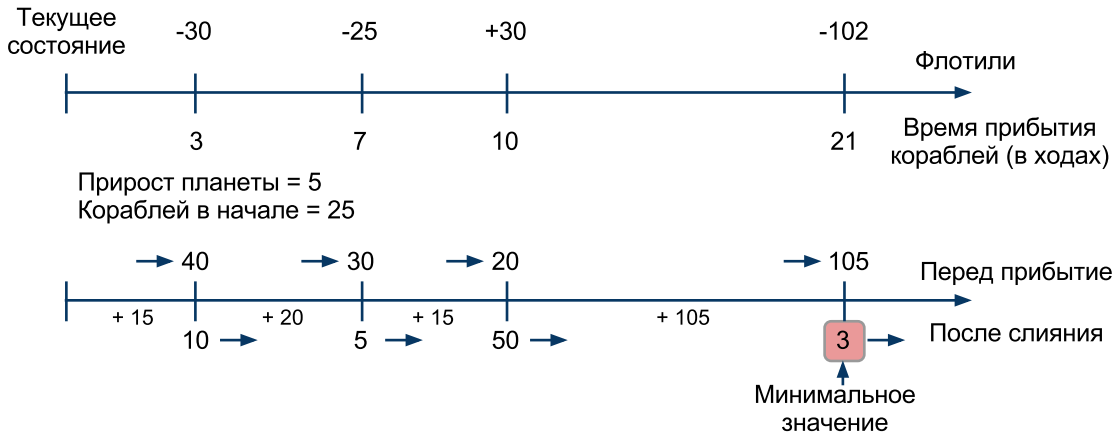


Рис. 2.6: График работы симуляции прибывающих кораблей.

На верхней шкале показано будущее планеты с изначально 25-ью кораблями и подлетающими 4 флотилиями (чёрточка на отрезке), 3 из которых чужие (со знаком минус) и одна своя (со знаком плюс). На нижней шкале между чёрточками показано сколько планета успевает набрать (со знаком плюс) между прилётами кораблей; верхние цифры показывают сколько кораблей было к момент прилёта флотилии; снизу показано сколько кораблей осталось после вычитания (или сложение) с прилетевшей флотилией.

Прирост у планеты составляет 5 кораблей, следовательно к прилёту первой флотилии на планете уже будет 40 кораблей. Корабли противника вычитаются и на планете остаётся 10 кораблей (под шкалой) и так далее.

Для принятия решения о том сколько кораблей необходимо забронировать для защиты надо найти минимальное количество кораблей, какое будет в будущем у планеты. В данному случае это число 3. Следовательно, что бы к этому моменту планета осталась наша (пусть даже без кораблей) сейчас на ней должно остаться $25 - 3 = 22$ корабля, а остальными уже можно пользоваться.

> “Стратегия ведения войны такова: не полагайся на то, что враг не придет, полагайся на средства, которыми располагаешь, чтобы принять его. Не полагайся на то, что враг не нападет; полагайся на то, чтобы наши позиции были неуязвимы для нападения.
> …
> С теми, кто рядом, ожидай далекого; с отдохнувшими ожидай усталого; с сытыми ожидай голодного. В этом путь управления силой.”
> (Tzu, 6 век до Н.Э.)

Что бы максимально обезопасить себя от захвата планеты можно посмотреть на ближайшие планеты противника и предположить, что они на следующем ходу выпустят

всё что у них есть по направлению к данной планете. Потенциальной угрозой можно считать все планеты противника, что находятся ближе чем первая ближайшая своя планеты. Это показано на рисунке 2.7.

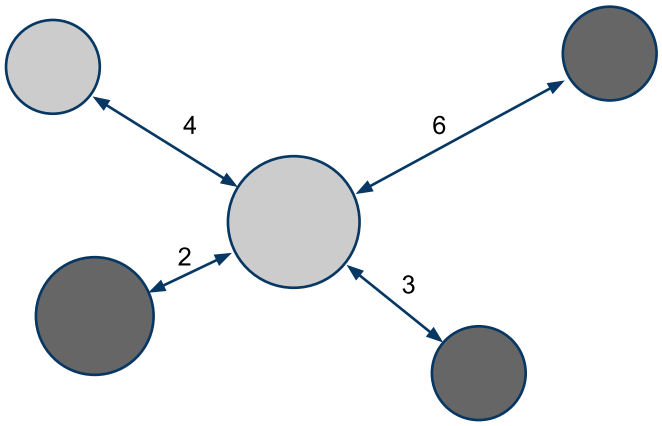


Рис. 2.7: Учёт потенциальной угрозы со стороны ближайших планет противника.

На рисунке видно, что потенциальной угрозой можно считать две нижних планеты, верхняя же планета противника (тёмная) не является угрозой, т.к. на подмогу центральной успеет прийти своя же планета (сверху светло-серая).

Бронирование кораблей необходимо делать в отдельном цикле так, что бы все другие этапы уже имели информацию о том, на сколько кораблей они могут рассчитывать.

Теперь должно быть понятно, почему использование особой логике на первом шагу не обязательно. В случае первого шага данный алгоритм самозащиты увидит стоящую рядом планету противника и рассчитает сколько ему можно пустить на расширение так, что бы в случае если все 100 кораблей полетят на него, он бы смог отбиться.

2.3.3 Защита своих войск

> “Непобедимость заключена в самом себе; возможность победы зависит от врага.
> …
> Поэтому сказано, что стратегию победы над врагом можно познать, но не всегда можно применить.”
> (Tzu, 6 век до Н.Э.)

Написать идеальные алгоритм бронирования кораблей для самозащиты невозможно. Всегда будет вероятность при которой, симулируя исход для планеты, окажется, что планета всё таки переходит к противнику. В случае, когда из симуляции следует, что

в обозримом будущем планета перейдёт к противнику, то в таком случае необходимо взять все свои ближайшие планеты и отсортировав их по мере удаления от неё. После отправить с каждой по мере возможности столько кораблей, сколько нужно, что бы в конце планеты осталась у прежнего игрока.

2.3.4 Атака

Когда все необходимые корабли забронированы для самообороны, и все флоты высланы на подмогу своим планетам, тогда все оставшиеся свободные корабли можно использовать для атаки. Краеугольным камнем атаки является правильный выбор цели. Для этого необходимо написать оценочную функцию, которая будет оценивать полезность атаки не своих планет для каждой своей планеты. Таким образом у каждой своей планеты образуется список планет отсортированный по этому коэффициенту, из которых каждая по одной рассматривается в качестве цели. В случае, если количество кораблей, которое будет на планете цели к моменту прилёта флотилии будет меньшим, чем есть на данной (для которой ищется цель), то можно отправить флотилию для её захвата.

Например: Планета противника находиться на расстоянии 3 шагов с 10-ью кораблями и приростом в 2 планеты. Таким образом для того, что бы её захватить необходимо выслать $10 + (3 * 2) + 1 = 17$ кораблей. Если на планете игрока нет такого количества, то в качестве цели рассматривается следующая не своя планета.

Важно понимать, что отправлять флотилию для атаки можно только будучи уверенном в её захвате, иначе игрок просто потеряет корабли, ничего не получив в замен.

Оценочную функцию можно занять из экономики, на основе которой появилась наука “Теория Игр”. Все не свои планеты можно представить в виде фондов или недвижимости, в общем всего того, что приносит деньги и является целью для инвестиций. Корабли при этом представляют собой деньги. В момент отсылки кораблей деньги тратятся. Находясь в полёте денег уже нет. Долетев до цели планета меняет владельца - происходит инвестирование средств. Через какое-то время (зависит от роста планеты) на планете возникнет сумма, которую мы изначально потратили. Это называется *окупаемостью инвестиций* (англ. “`Return on Investment, ROI`”). В случае игры она измеряется в ходах (т.е. за сколько ходов этот вклад само-окупиться) (см. формулу 2.1, стр. 25).

$$k_{pq} = d_{pq} + (\frac{S_q}{G_q}) \tag{2.1}$$

где
k_{pq} – коэффициент для планеты-цели q относительно планеты p
d_{pq} – расстояние в ходах между планетами p и q
S_q – количество кораблей на планете q
G_q – скорость прироста на планете q

Таким образом не всегда самые близкие и самые слабые планеты становятся более желаемой целью, как показано на рисунке 2.8.

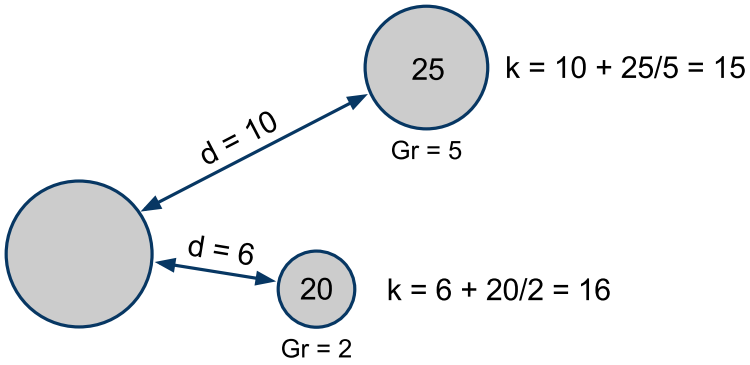


Рис. 2.8: Планета с коэффициентом = 15 является более приоритетной целью. (Здесь Gr - это скорость роста (англ. “`Growth Rate`”)

Эту функцию оценки нельзя использовать в чистом виде, т.к. могут возникнуть ситуации, когда планета с самым низким коэффициентом находится ближе к противнику чем к вам. В таком случае такая атака может провалиться. Причиной тому хитрость под названием “перехват”, которая будет рассмотрена ниже (см. 2.3.6). А пока можно просто ввести как правило, что атаковать планеты, которые ближе к противнику чем к игроку - нельзя. Особенно это касается первых ходов, когда планеты с хорошим коэффициентом находятся как на одной так и на другой “сторонах поля”. Так же нежелательны любые длинные перелёты кораблей, т.к. это ведёт к потери контроля над ситуацией, и за время полёта флотилии противник успеет сгруппироваться и защититься. Для этого в оценочную функцию можно внести зависимость от расстояние. Например, умножение расстояние на некий коэффициент больше нуля, таким образом заставляя увеличивать результат функции по мере увеличения расстояния.

2.3.5 Перегруппировка

Как и инвесторы не любят, когда их деньги лежат и “не работают”, так и игрок должен заботиться о том, что бы все генерируемые корабли приносили свой вклад в победу. В частности может возникнуть ситуация, когда рядом с данной планетой нету ни одной

нейтральной или планеты противника. В таком случае все свободные корабли должны быть переправлены ближе к “линии фронта”, которая часто образуется в игре между достаточно развитыми ботами.

> “Если мы можем идти вперед и противник тоже может продвигаться, это называется “доступной местностью”. На такой местности, первым делом занимай высоты и [строну] ян, а также обеспечь пути для подвоза провианта. Тогда, если мы вступим в битву, за нами будет преимущество.”(Tzu, 6 век до Н.Э.)

Что бы перегруппировать корабли необходимо найти цель, которую надо атаковать, после чего найти ближайшую к нему свою планету. Эта планета называется *атакующей*. Зная атакующую планету необходимо найти к ней наиболее подходящий путь. Для этого можно представить все планеты в виде *полносвязного графа*, где *вершинами* являются планеты, а все возможные пути между ними *рёбрами*. Тогда можно воспользоваться одним из имеющихся алгоритмов поиска кратчайшего пути между вершинами. (Cormen, 2001)

Но для упрощения задачи можно найти такой путь более простым способом. Он схематично изображён на рисунке 2.9.

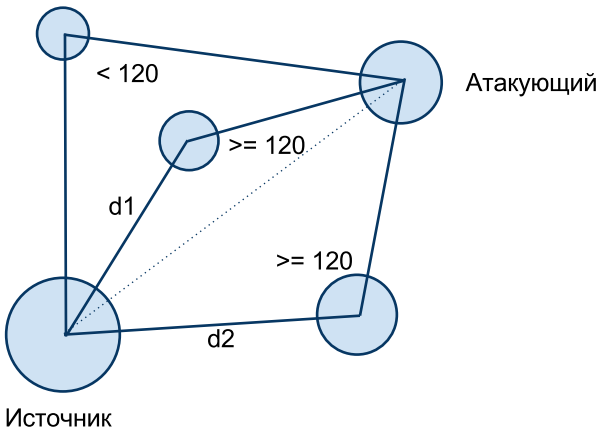


Рис. 2.9: Выбор потенциального звена при перегруппировки сил по направлению к атакующему

Алгоритм заключается в следующем. Берутся все планеты игрока и сортируются по расстоянию до атакующего. По одному проверяется угол между планетой, кандидатом и атакующем. Если угол больше или равен 120^o, значит это и есть искомое звено, куда надо отправить свободные корабли. Данный угол был получен опытным путём. Объяснить такой выбор можно тем, что это достаточно тупой угол, и при нём не создаётся слишком длинных маршрутов. Таким образом каждая планета отвечает только за правильную доставку до следующего звена.

2.3.6 Хитрость и тактические приёмы

> “Война - это путь обмана. Поэтому, даже если [ты] способен, показывай противнику свою неспособность. Когда должен ввести в бой свои силы, притворись бездеятельным. Когда [цель] близко, показывай, будто она далеко; когда же она действительно далеко, создавай впечатление, что она близко.”
> (Tzu, 6 век до Н.Э.)

В игре ВП возможно применять так же и некоторые хитрости и тактические приёмы.

Одна такая хитрость называется “перехват”. Суть её в том, что бы перехватывать только что захваченные врагом планеты. Т.к. чаще всего игроки захватывают планеты с минимальным количеством кораблей во флотилии, то на планетах, сразу после захвата практически нет кораблей. Если такой случай вовремя предвидеть, и вовремя послать свои корабли, прилетающие туда же на один ход позже, то можно небольшой жертвой захватить новую планету.

Другая хитрость касается эффективной защиты от нападения. Если противник высылает флотилию достаточную для захвата планеты игрока, то в таком случае если игрок тут же вышлет подмогу, это сразу может увидеть противник и дослать ещё кораблей. Такая череда дополнительных флотилий со стороны подмоги и досылки для атаки будет постоянной. Тут можно сделать следующее - необходимо дождаться такого момента высылки подмоги, когда она придёт точно в тот же момент, что и атакующая флотилия. Таким образом можно выиграть время, т.к. противник увидит, что кораблей для захвата не хватает только в самый последний момент.

Самооборону противника можно ослепить, если выслать атакующий флот не на прямую на планету противника, а на соседнюю от неё нейтральную, а по прибытию на неё тут же выслать все корабли на противника. Простые алгоритмы проверяют лишь непосредственную угрозу, а в такой ситуации она будет незамеченной и противник не успеет подготовиться достаточно кораблей для обороны.

Важно отметить, что от каждой придуманной тактики и хитрости должна быть продумана и реализована эффективная защита.

2.4 Внутреннее строение бота

Внутренняя структура программы бота дублирует стратегический план, описанный выше, разбивая процесс принятия решения на отдельные блоки (рис. 2.10).

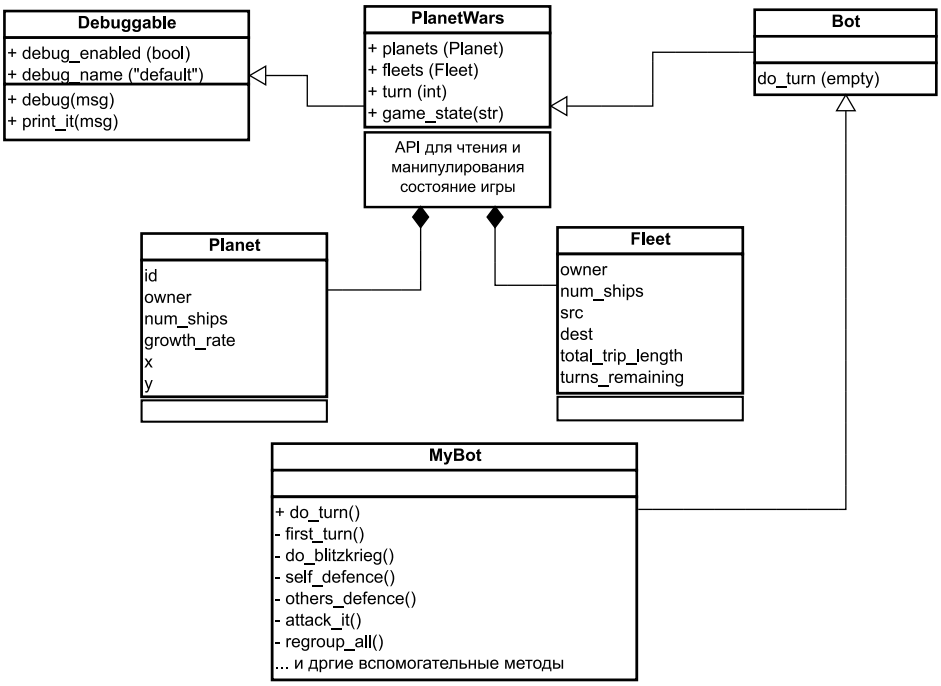


Рис. 2.10: Упрощённая архитектура классов наследования бота

На рисунке видно, что класс с самим ботом `MyBot` наследуется от класса `Bot`, который является подобием интерфейса (в Python нету понятия интерфейса как такового). `Bot` наследуется напрямую от класса `PlanetWars`, хранящего состояние игры и все необходимые API для работы с ним: методы восприятие среды и воздействие на неё. В частности, в `PlanetWars` хранится список планет и список флотилий. Сам `PlanetWars` наследуется от класса `Debbugable`, который отвечает за обеспечение условного журналирования и вывода на экран отладочных данных. Пример исходного кода приведён в листинге 2.1.

```
1  def sim_arrivals(self, distances, p):
2      """Simulate end state of the game for given planet.
3
4          distances: List of tuples of objects are going to this planet.
5          p: planet for simulation
6      """
7      history = []
8      prev_distance = 0
9      history.append((p.owner, p.num_ships, 0))
10     for dist_group in distances:
11         diff_distance = dist_group[0][1] - prev_distance
```

```
12          if p.owner != 0:
13              p.num_ships += p.growth_rate * diff_distance #next group came
14          prev_distance = dist_group[0][1]
15
16          participants = {p.owner:p.num_ships}
17          for obj in dist_group:
18              if not obj[0] in participants:
19                  participants[obj[0]] = obj[2]
20              else:
21                  participants[obj[0]] += obj[2]
22
23          winner = (0, 0)
24          second = (0, 0)
25          for k, v in participants.items():
26              if v > second[1]:
27                  if v > winner[1]:
28                      second = winner
29                      winner = (k, v)
30                  else:
31                      second = (k, v)
32
33          if winner[1] > second[1]:
34              p.num_ships = winner[1] - second[1]
35              p.owner = winner[0]
36          else:
37              p.num_ships = 0
38          history.append((p.owner, p.num_ships, dist_group[0][1]))
39      return history
```

Listing 2.1: Метод симулирующий прибытия флотилий

В листинге приведёт метод, который на вход принимает список объектов, имеющих время прибытие, количество кораблей и принадлежность. В этом списке могут быть как флотилии так и планеты противника представляющие потенциальную угрозу. На строке 10 эти объекты итерируются, где каждый проходит ряд изменений и вычислений, идентичных тем, что производит игровой движок, когда во время применения решений игроков (рис. , стр.) считает итоге прилётов флотилий на планеты. А именно: строки 12-14 икрементирует рост планеты, 16-21 разбивает по группам всех участников локального боя (какой игрок сколько кораблей привносит), 23-31 рассчитывают победителя и проигравшего (тот кому достанется планета), 33-37 вычитает корабли и меняет владельца планеты, 38 добавляет результат от прилёта флотилии(ий) в результативный список.

На данный момент реализованы все основные механизмы игры, за исключением хитростей и тактичесих манёвров. Автор планирует закончить до окончания соревнования.

ЗАКЛЮЧЕНИЕ И ВЫВОДЫ

Результатом исследования автора стали более глубокие и широкие познания в области искусственного интеллекта. Прочитанные книги, статьи, просмотренные выступления специалистов на тему искусственного интеллекта были не бесполезны лично автору. Он познакомился с этой областью науки, что являлось одной из целей этой работы.

Полученные знания помогли автору в написании практического задания и участие в конкурсе “Google AI Challenge”. Изученный материал позволил более чётко представить себе все возможные направления при написании ИИ. Были рассмотрены основные направления при написании ИИ и классификация сред, в которых функционируют ИИ. По предложенной классификации были рассмотрены характеристики конкретной игры. Были сделаны предположения о неэффективности в данном случае более современных методов создания ИИ.

Всё это помогло автору выбрать направление, наиболее простое, но тем не менее по мнение автора эффективное. Так это или не так покажет время, но на данный момент возможности ИИ автора не предвещают полной неудачи на конкурсе, что говорит о жизнеспособности выбранного пути написания ИИ. Профиль автора и все сыгранные его ботом игры можно найти по адресу `http://ai-contest.com/profile.php?user_id=3984`

На момент написания работы автор всё ещё занимается разработкой ИИ, т.к. соревнование заканчивается в декабре 2010 года. На данный момент более 1000 строк кода написано и потрачено более 100 человеко-часов. По условиях конкурса и исходя из здравого смысла выкладывать код самого ИИ нельзя, по-этому его нет на сайте github. Тем не менее полный исходный код бота можно найти по адресу `http://soswow.pri.ee/ai/mybot.zip`.

Все дополнительные задания по написанию собственного игрового движка, доработке стандартного визуализатора, созданию процесса инкрементальной разработки ИИ были выполнены и безусловно помогли автору в написании и улучшении бота. Переписанный автором движок на языке Python код можно найти на сайте git-hub по адресу `http://github.com/soswow/python-ai-challenge`.

При изучении древнего трактата о военном ремесле Сунь Дзыня “Искусство войны” автор обнаружил несколько подходящих под описание стратегии игры мыслей. Пусть

игра называется “Войны планет”, всё же нельзя говорить, что описанные советы и законы могут один в один лечь в основу стратегии ИИ. Впрочем это и ожидаемо, игра слишком проста, что бы в ней можно было применить всю глубину мудрости описанного в трактате.

Написание работы на LaTeXстало не таким простым занятием. Основная проблема, с которой пришлось столкнуться автору - это обеспечение всех официальных и необходимых требований к оформлению работы. С большей частью удалось справиться, но некоторые недочёты останутся по всей видимости до следующей работы. Исходный код данной курсовой работы также находиться на GitHub’е по адресу `https://github.com/soswow/ai-paper`.

A ПРИЛОЖЕНИЕ. ОТЧЁТ ПО КУРСОВОЙ ПРАКТИКЕ

-

-

-

-

Литература

AIChallangeFAQ. Google ai challenge faq, 10 2010. URL http://ai-contest.com/faq.php.

M. Alan. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

M. Buckland. *Programming game AI by example*. Wordware, 2005. ISBN 1556220782.

T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001. ISBN 0262032937.

GoGameWiki. Go, 2010. URL http://en.wikipedia.org/wiki/Go_game.

S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1994. ISBN 0023527617.

Soren Johnson. Playing to lose: Ai and "civilization", 09 2010. URL http://www.youtube.com/watch?v=IJcuQQ1eWWI.

W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in neural nets. *Bulletin of Mathematical Biophysics*, 5(1):15–137, 1943.

I. Millington and J. Funge. *Artificial intelligence for games*. Morgan Kaufmann, 2009.

H. Moravec. When will computer hardware match the human brain. *Journal of Evolution and Technology*, 1(1), 1998.

O. Morgenstern and J. Von Neumann. *Theory of games and economic behavior*. Princeton University Press Princeton, NJ, 1947.

PlanetWarsSpec. Planet wars specification, 10 2010. URL http://www.ai-contest.com/specification.php.

S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall, 1995.

J.R. Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(03):417–424, 1980.

S. Tzu. *The art of war*. 6 век до Н.Э.

НГ Рамбиди, ЕП Гребенников, АИ Адамацкий, АГ Девятков, and ДВ Яковенчук. *Биомолекулярные нейросетевые устройства*. Радиотехника М, 2002.