Title page removed for privacy reasons

Оглавление

Аннотация

Файтинги - сложный жанр игр, привлекающий особый вид игроков, которые любят сложные матчи и часто предпочитают тренироваться против реальных игроков. К сожалению, некоторые игры имеют небольшие сообщества, что делает поиск онлайн-соперников проблематичным. Данная работа предлагает решение и объясняет реализацию бота для файтинг-игры ”Acceleration of SUGURI 2”. Система была разработана с учётом закрытого исходного кода игры. В данном случае компонент компьютерного зрения определяет производительность системы с точки зрения вычислений, а также качества игрового процесса, тем самым доказывая, что агенты, основанные на обучении с подкреплением, могут использовать компьютерное зрение для сбора наблюдений. Наконец, в данная работа объясняет причины неудач проекта в руках одного неопытного разработчика и предлагает возможные улучшения и области дополнительных исследований.

Глава 1

Введение

Видеоигры привлекают повышенное внимание людей благодаря развитию Интернета. Сегодня люди могут играть в различные игры вместе в режиме онлайн. В то время как некоторые жанры доминируют на рынке, другие остаются непопулярными. Например, файтинги - это нишевый жанр с уникальными отличиями геймплея. Эти особенности делают файтинги более сложными для освоения в сравнение с остальными жанрами. С другой стороны, сложность обычно привлекает игроков, готовых потратить множество часов на изучение тонкостей игры.

Особенность файтингов в том, что они работают сразу на трёх уровнях игры. На первом уровне, игрок должен сконцентрироваться на правильном исполнении атак и комбинаций, и некоторые из них могут требовать высокую точность нажатий вплоть до одного кадра. После изучения первого уровня, игрок начинает задумываться о более эффективной стратегии использования изученных движений для больших шансов на победу в сражении с тем или иным персонажем. нНа этом уровне важно не просто правильно выполнять комбинации, но и думать о том, в какой ситуации лучше

принимать какие решения, т.е. активно реагировать на происходящее. На третьем же уровне игрок задумывается о действиях другого игрока, пытаясь предугадать его решения и обмануть его с целью застать врасплох или поймать на ошибке.

Данные особенности игр нравятся далеко не всем игрокам, вследствие чего у большинства файтингов наблюдается очень маленькая база игроков. Среди тех, кто уделяет своё время игре, могут быть игроки из разных стран и часовых поясов, что заметно усложняет поиск противников в онлайне. Более того, разный уровень игры очень сильно усложняет игровой опыт для новых игроков.

С одной стороны, файтинги обычно предлагают встроенных ботов в одиночной игре. Такие боты позволяют ознакомиться с основами игрового процесса, но перестают представлять какую-либо угрозу для тех, кто освоился в игре. Таким образом, тренировка против таких ботов не даст таких результатов, как тренировка против людей в онлайне.

Acceleration of SUGURI 2 (AoS2) [1] - это смесь файтинга и shoot-them-up жанров. Вместо привычной игры с блоками, игроки должны стрелять множеством снарядов и уклоняться от них. Игра предоставляет рывки как более быстрый способ передвижения, мощные атаки ценой особой энергии, а шкалу нагрева, при высоких значениях которой получаемый урон заметно увеличивается.

AoS2 так же страдает от основных проблем файтинг игр - маленькое сообщесво и сильная разница в уровне игры. Для решения данной проблемы.

Данная работа предлагает возможное решение данной проблемы - бот основанный на машинном обучении, который позволил бы игрокам трени-

Рис. 1.1: Игровой процесс Acceleration of SUGURI 2

роваться против сильного соперника офлайн.

К сожалению, существует мало игровых ботов, которые объединяют машинное обучение и компьютерное зрение. В случае с AoS2, компьютерное зрение кажется единственным решением, поскольку у игры закрытый исходный код и отсутствует интерфейс (API) для доступа к памяти игры во время работы приложения.

Таким образом, главная цель проекта - доказать, что совмещение компьютерного зрения и машинного обучения для создания бота может быть удачным решением. К сожалению, не получится создать продвинутого бота из-за достаточно высокой сложности проекта. Тем не менее, можно разработать скелет системы, которая позволит добиться лучших результатов путём независимого улучшения отдельных компонентов.

Глава 2

Обзор литературы

Цель поиска литературы состояла в том, чтобы выяснить, какие методы компьютерного зрения и машинного обучения лучше всего подходят для данного проекта. Также, в литературу включены работы по другим проектам из сферы ботов для видеоигр.

Машинное Обучение (ML) [2] позволяет находить зависимости во входных данных и предсказывать какие-либо метрики для новых данных. Вся область ML делатся на три парадигмы [3] - обучение с учителем, обучение без учителя, и обучение с подкреплением. Обучение с учителем требует заранее указывать формат данных и желаемые выходные данные, а обучение без учителя находит зависимости в неструктурированных данных. Обучение с подкреплением симулирует прямое взаимодействие агента со средой и обучение на ошибках. Таким образом, обучение с подкреплением больше всего подходит к текущей задаче, так как в качестве агентов можно представить игровых персонажей, а в качестве среды - игровую арену.

Существуют различные методы Компьютерного зрение, которые позволят взаимодействовать с игрой. Сравнение по шаблону (Template

Matching) [4] пытается найти прямые вхождения какого-либо шаблона в изображении, но не справляется с поворотами и масштабированием. Каскадные классификаторы (Cascade Classifier) [5] могут распознавать объекты по цветам и форме по принципе от меньшего к большему. Такие классификаторы можно натренировать [6] на распознавание одного типа объектов с разными поворотами и размерами. Поскольку прошлые решения не могут находить разные типы объектов, самым подходящим решением будет You Only Look Once (YOLO) [7] классификатор, который позволяет за один проход находить разные классы объектов с поворотами и масштабированием.

Игровые боты на машинном обучении - довольно непопулярная тема. Тем не менее, среди ярких примеров есть боты для DOOM [8] и Dota 2 [9]. В обоих случаях использовались интересные подходы и оптимизации, такие как алгоритм ”разделяй и властвуй пропуск кадров для оптимизации времени тренировки, длинная кратковременная память (Long Short-Term Memory, LSTM) [10], и другие. К сожалению, эти проекты не используют методы компьютерного зрения. Также, как отдельный класс проектов существуют боты для игр на Atari [11], которые учатся напрямую из необработанного изображения на экране, что в теории может избавить от потребности в использовании компьютерного зрения.

Таким образом, обзор литературы становится хорошей отправной точкой для начала работы над проектом, поскольку теперь более понятны требования и ограничения различных методов.

Глава 3

Методология

3.1 Общая архитектура

Посколько исходный код игры закрытый, нет простого способа напрямую взаимодействовать с игрой во время работы. На Рис. 3.1 показаны основные компоненты системы. Компонент `Environment` представляет из себя адаптер [12] между окном игры и агентом.

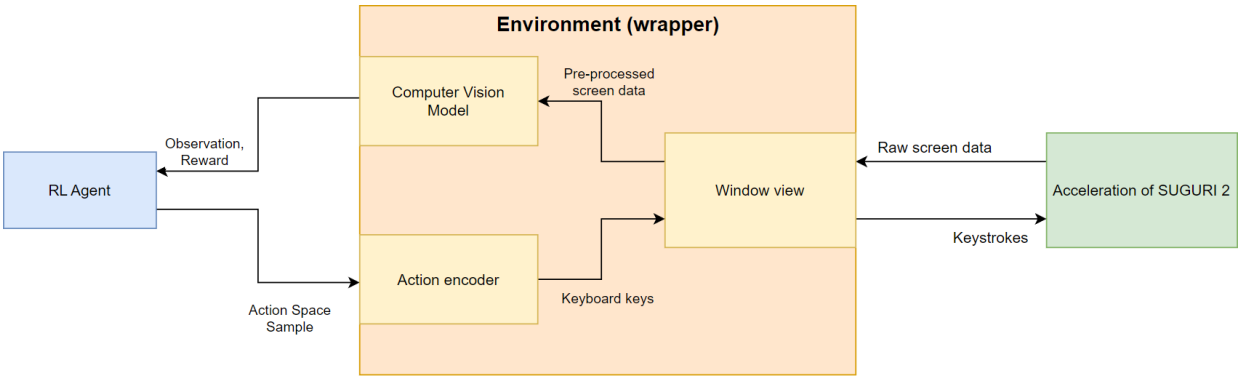

Рис. 3.1: Общая архитектура проекта

`Environment` содержит в себе следующие компоненты. `Window View` отвечает за прямое взаимодействие с игрой - симуляцию нажатий клавиш, а также чтение отрисованного кадра с экрана. `Computer Vision`

`Model` обрабатывает полученные кадры и извлекает из них данные. `Action Encoder` преобразует данные о желаемых действиях агента в клавиши, которые должны быть нажаты. Таким образом, `Environment` конвертирует и пересылает данные между компонентами системы.

3.2 Работа с изображением

Рис. 3.2 показывает, что игра предоставляет режим высокой контрастности, в котором фон становится серым, а персонажи и их атаки ярко помечаются цветом.

Наконец, модуль компьютерного зрения должен извлекать данные из верхнего интерфейса, показанного на Рис. 3.3. На данный момент важны только следующие части интерфейса.

1. **Текущее здоровье (Health Points, HP)** и **Красное Здоровье (Heat Damage)**
2. **Шкала энергии**
3. **Шкала нагрева**
4. **Время** до конца раунда.

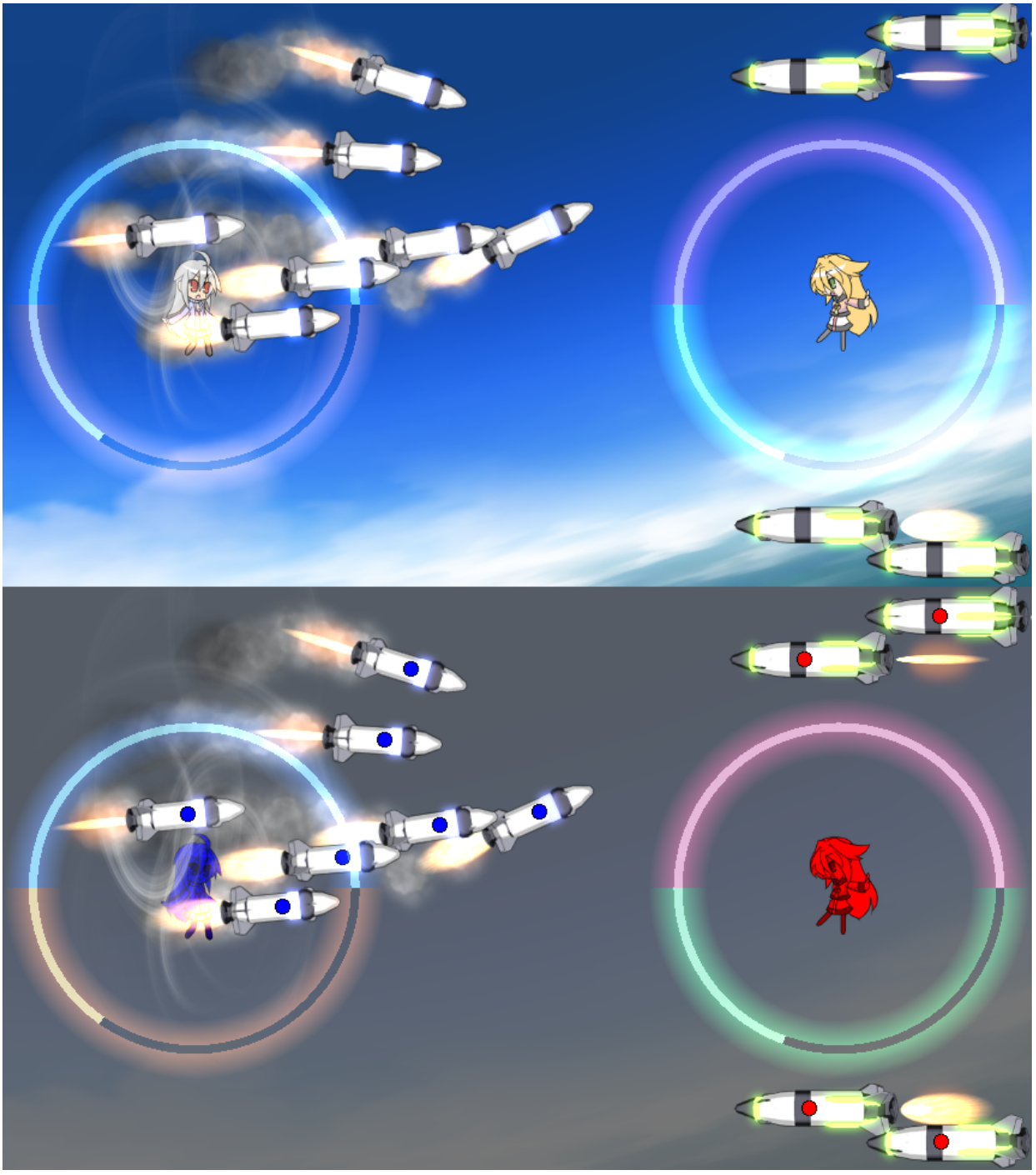

Рис. 3.2: Сравнение нормального режима и режима высокой контрастности

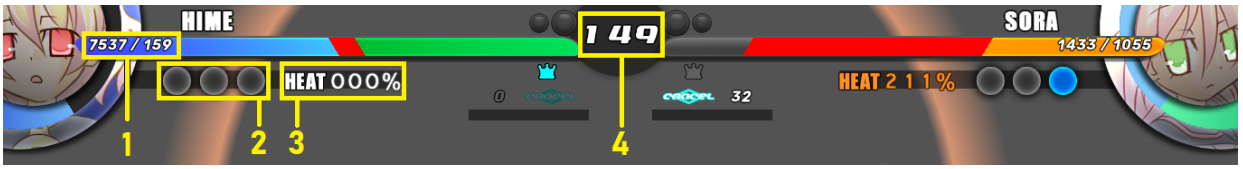

Рис. 3.3: Верхний интерфейс во время сражения

Глава 4

Реализация

Языком реализации был выбран Python 3, потому что он предоставляет множество библиотек. Anaconda [13] была выбрана как основной менеджер пакетов из-за удобства установки тяжёлых библиотек.

4.1 Модификация игры

Чтобы улучшить точность модулей компьютерного зрения, были использованы инструменты от сообщества игры [14], чтобы заменить внутриигровые текстуры. Таким образом, были удалены некоторые частицы и эффекты, а также полностью заменены фоны на статичные чёрный цвет. Данные изменения должны сделать обнаружение объектов более точным.

4.2 Windows API для взаимодействия с игрой

Поскольку игра официально поддерживает только OS Windows, пришлось использовать Windows API через библиотеку `pywin32`. Для взаимо-

действия игры требуется один раз найти хэндл (handle) окна - уникальный идентификатор, который не меняется на протяжении работы приложения. Таким образом, было настроено чтение кадров напрямую из окна игры.

Поскольку игра использует DirectInput для обработки нажатий, сторонняя библиотека `pydirectinput_rgx` [15] занималась симуляцией нажатий. К сожалению, нажатия отрабатывают глобально во всей системе, поэтому был разработан модуль, полностью отключающий симуляцию нажатий когда окно игры неактивно.

4.3 Обработка интерфейса

Большую часть верхнего интерфейса статична. Таким образом, в случае с иконками достаточно читать один пиксель, и по его значению определять, активна ли иконка или нет.

К сожалению, нагрев, здоровье, и таймер представляют из себя текст, который простым способом читать не получится. По этой причине использовалась библиотека `tesserocr` [16], напрямую использующая API инструмента для распознавания текста Tesseract OCR [17].

Были использованы разные методики обработки изображения перед распознаванием текста - обесцвечивание, обрезание, изменение значений в зависимости от заданного порога, и удаление шума. Все эти методы предоставляются библиотекой OpenCV и значительно улучшают точность распознавания. Более того, была введена простая система исправления ошибок распознавания для замены похожих символов на цифры. Например, буквы `s` и `o` заменялись на цифры 5 и 0 соответственно.

4.4 Обнаружение объектов на арене

Самый подходящим решением для обнаружения объектов была модель YOLOv5 [18]. Для тренировки модели был подготовлен набор данных - кадров из игры. На этих изображениях запечатлён матч между игроком, который только уклонялся от атак, против компьютера, который пытался использовать как можно больше снарядов. Набор данных состоял 200 размеченных изображений. 170 изображений использовалось для тренировки, а валидация проходила на оставшихся 30.

Результаты тренировок показаны на Рис. 4.1. Из матрицы можно понять, что набор данных был достаточно неоднородным, что привело к низкому качеству результатов.

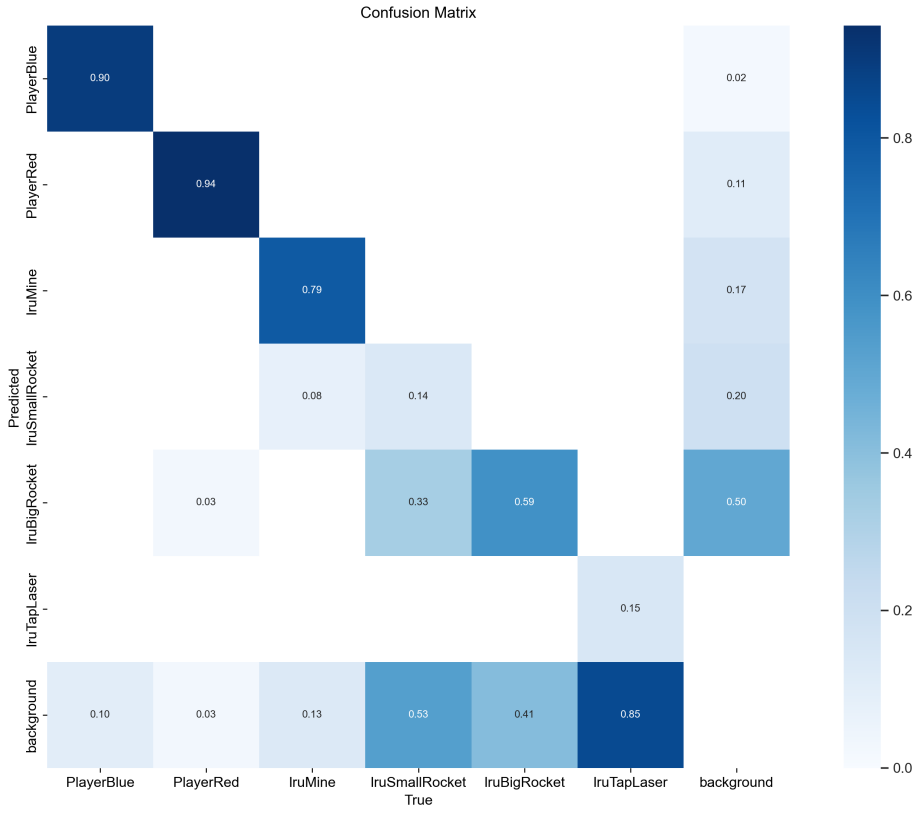

Рис. 4.1: Матрица неточностей

Натренированнаая модель была загружена через библиотеку PyTorch [19]. Тестирование показало довольно низкие результаты по следующим причинам: малый объём данных, несбалансированные классы по количеству объектов, а также шум на изображенияк в виде дополнительных частиц, которые не удалось убрать. Тем не менее, в некоторых случаях обнаружение работало достаточно успешно, как в примере на Рис. 4.2.

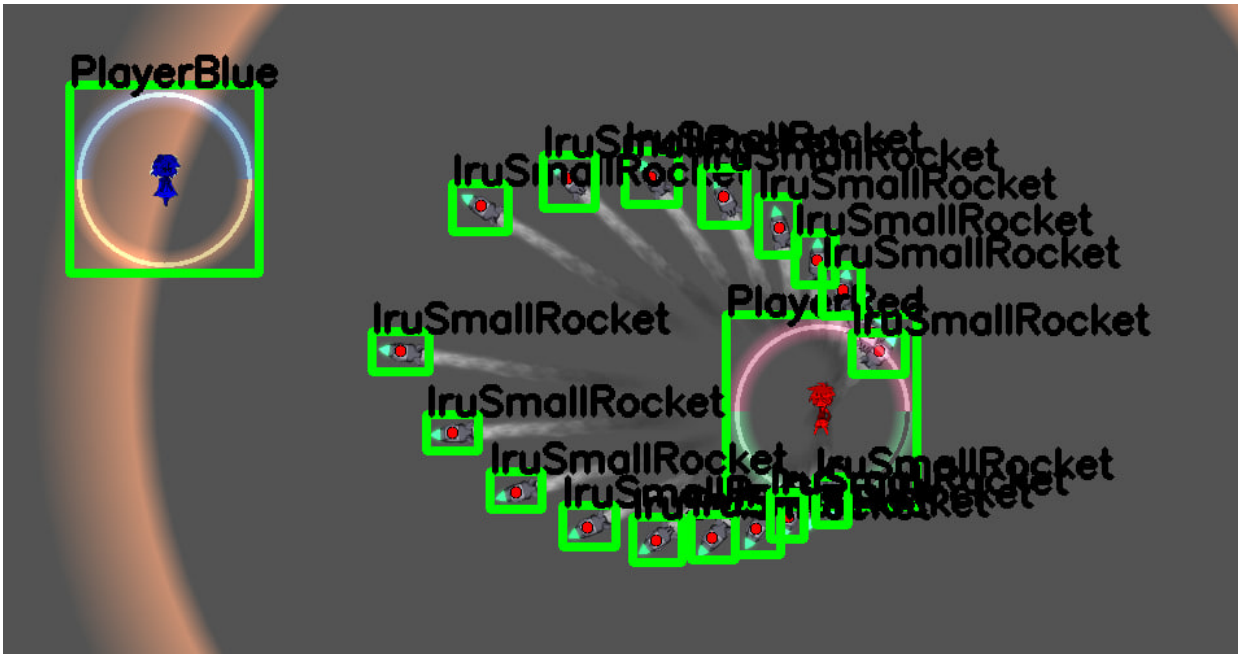

Рис. 4.2: YOLO обнаруживает большинство объектов

4.5 Тренировка агента

Из-за неудачных результатов с обнаружением объектов, запускать тренировку агента было бессмысленно. Тем не менее, был разработан скелет системы, который позволяет запустить тренировку.

Библиотека OpenAI Gym [20] позволяет описывать любую среду для RL в универсальном формате через реализацию класса `Environment` и

следующих методов: `reset()` перезапускает симуляцию и возвращает первое наблюдение, `step(action)` выполняет действие агента и возвращает новое наблюдение и награду, и `render()` - выводит текущее состояние среды на экран.

Также, среда должна описывать формат пространства действий агента и наблюдений. Действия агента - это список булевых значений, каждое из которых отвечает за конкретную клавишу. Наблюдения - хранилище значений по ключу. Значение включают в себя данные с верхнего интерфейса, а также все обнаруженные объекты на экране.

Простейшая система наград зависела только от полученного урона. Этот элемент системы сильно ограничен возможностями модуля компьютерного зрения, то есть обранужение большего количества сложных сценариев позволит более тонко настраивать систему награждения.

Тренировку можно осуществлять используя библиотеку Stable Baselines 3 [21]. Это библиотека предлагает набор из различных алгоритмов [22], и выбор зависит пространства действий агента, а также желаемого результата.

Глава 5

Выводы

Проект достиг минимальной цели - было доказано, что RL агент может получать данные, исопользуя компьютерное зрение. К сожалению, проект был слишком амбициозным для одного разработчика. Более того, сложность проекта была достаточно высокой, и на каждом шаге реализации встречались проблемы, каждая их которых может быть достойна отдельной исследовательской работы.

Низкая точность классификатора YOLO обусловлена малым набором данных. Помимо увеличения размера данных нужно заботиться о качестве разметки и количестве данных (рекомендуется хотя бы 1000 изображений для каждого класса). Также, можно перенести часть вычислений на графический процессор для ускорения обнаружения.

Тренировка RL агента в таких условиях оказалась бессмысленной. Даже при условии идеального обнаружения объектов, тренировка могла бы оказаться слишком долгой, поскольку игра не может обрабатывать более 60 кадров в секунду. Возможно, процесс можно было бы ускорить, используя трансферное обучение без отрисовки графики.

С другой стороны, система получилась достаточно универсальной и модульной, так как можно по отдельности улучшать существующие компоненты для улучшения общего результата.

Список использованной литературы

[1] Orange_Juice, *Acceleration of SUGURI 2*, [Video Game for PC], Mar. 2018. [Online]. Available: https://store.steampowered.com/app/390710/Acceleration_of_SUGURI_2/.

[2] T. M. Mitchell, *Machine Learning* (McGraw-Hill series in computer science). New York: McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[3] A. S. Lampropoulos and G. A. Tsihrintzis, *Machine Learning Paradigms* (Intelligent Systems Reference Library). Cham: Springer International Publishing, 2015, vol. 92, ISBN: 978-3-319-19134-8 978-3-319-19135-5. DOI: 10.1007/978-3-319-19135-5. [Online]. Available: http://link.springer.com/10.1007/978-3-319-19135-5.

[4] R. Brunelli, *Template matching techniques in computer vision: theory and practice*. Chichester, U.K: Wiley, 2009, OCLC: ocn289008992, ISBN: 978-0-470-51706-2.

[5] P. Viola and M. Jones, «Rapid object detection using a boosted cascade of simple features», in *CVPR 2001*, vol. 1, Kauai, HI, USA: IEEE Comput. Soc, 2001, pp. I–511–I–518, ISBN: 978-0-7695-1272-3. DOI: 10.1109/

CVPR.2001.990517. [Online]. Available: http://ieeexplore.ieee.org/document/990517/.

[6] OpenCV. «Cascade classifier training». Accessed: Feb. 14, 2023. (), [Online]. Available: https://docs.opencv.org/4.5.5/dc/d88/tutorial_traincascade.html.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», 2015, Publisher: arXiv Version Number: 5. DOI: 10.48550/ARXIV.1506.02640. [Online]. Available: https://arxiv.org/abs/1506.02640.

[8] G. Lample and D. S. Chaplot, «Playing FPS Games with Deep Reinforcement Learning», 2016, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1609.05521. [Online]. Available: https://arxiv.org/abs/1609.05521 (visited on 10/20/2022).

[9] OpenAI, : C. Berner, *et al.*, *Dota 2 with large scale deep reinforcement learning*, 2019. DOI: 10.48550/ARXIV.1912.06680. [Online]. Available: https://arxiv.org/abs/1912.06680.

[10] S. Hochreiter and J. Schmidhuber, «Long short-term memory», *Neural Comput.*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing Atari with Deep Reinforcement Learning*, 2013. DOI: 10.48550/ARXIV.1312.5602. [Online]. Available: https://arxiv.org/abs/1312.5602.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley

Professional Computing Series). Pearson Education, 1994, ISBN: 9780321700698. [Online]. Available: https://books.google.bg/books?id=6oHuKQe3TjQC.

[13] A. Inc., *Anaconda - Where packages, notebooks, projects and environments are shared*, Accessed: Mar. 3, 2023. [Online]. Available: https://anaconda.org/.

[14] M. Jarjour, *Acceleration of Suguri 2 Data Ripper*, Accessed: Feb. 15, 2023. [Online]. Available: https://github.com/MergeCommits/aos2ripper.

[15] *Python mouse and keyboard input automation for Windows using Direct Input*, Accessed: Mar. 3, 2023. [Online]. Available: https://github.com/reggx/pydirectinput_rgx.

[16] *A Python wrapper for the tesseract-ocr API*, Accessed: Mar. 3, 2023. [Online]. Available: https://github.com/sirfz/tesserocr.

[17] *Tesseract Open Source OCR Engine (main repository)*, Accessed: Mar. 3, 2023. [Online]. Available: https://github.com/tesseract-ocr/tesseract.

[18] Ultralytics, *YOLOv5*, Accessed: Feb. 14, 2023. [Online]. Available: https://github.com/ultralytics/yolov5.

[19] T. L. Foundation, *Pytorch*, Accessed: Mar. 3, 2023. [Online]. Available: https://pytorch.org/.

[20] G. Brockman, V. Cheung, L. Pettersson, *et al.*, «OpenAI Gym», 2016. DOI: 10.48550/ARXIV.1606.01540. [Online]. Available: https://arxiv.org/abs/1606.01540.

[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, «Stable-baselines3: Reliable reinforcement learning implementations», *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: `http://jmlr.org/papers/v22/20-1364.html`.

[22] Stable Baselines3, *RL Algorithms*, Accessed: Apr. 27, 2023. [Online]. Available: `https://stable-baselines3.readthedocs.io/en/master/guide/algos.html`.