Programistyczna gra robotów – algorytmiczne zawody AI z niepełną informacją

(Programming robot game – algorithmic AI competition with imperfect information)

Jacek Bizub Kamila Kubala

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

19 lutego 2022

Streszczenie

Praca ta opisuje projekt polegający na stworzeniu dwuosobowej gry programistycznej. Celem gry jest napisanie sztucznej inteligencji dla grupy robotów, która zdoła pokonać flotę przeciwnika. Gra została opublikowana na platformie CodinGame i umożliwia graczom rozwój umiejętności programistycznych i algorytmicznych. W pracy zostały opisane szczegółowe zasady gry i wymagania, które musi spełnić użytkownik. Dołączone zostały również szczegóły implementacji.

This thesis describes an implementation of a two-person programming game. The purpose of the game is to write artificial intelligence for a group of robots and the goal is to defeat the enemy fleet. The game is published on the CodinGame platform, allowing players to develop their programming and algorithmic skills. This work describes both the detailed rules of the game and the requirements that the users' code must meet. Implementation details are also included.

Spis treści

Rozdział 1.

Wstęp

1.1. Cel pracy

Niniejsza praca pisemna jest opisem i analizą projektu, mającego na celu stworzenie w pełni funkcjonalnej gry programistycznej **Clash of Bots**. Została ona opublikowana na platformie **CodinGame**[1] jako turniej dla dwóch graczy. Rozgrywka polega na kontrolowaniu grupy robotów tak, aby pokonać przeciwnika. W tym celu gracz musi napisać bota sterującego grupą w wybranym języku programowania. Program gracza powinien analizować sytuację na planszy i na tej podstawie podejmować decyzje o ruchu poszczególnych jednostek. Rozgrywka podzielona jest na maksymalnie 100 tur. W każdej turze jeden robot musi wykonać dokładnie jedną akcję.

1.2. Gry programistyczne

Historia gier komputerowych sięga aż do lat 50. dwudziestego wieku. Za jedną z pierwszych uznaje się implementację popularnej gry w **kółko i krzyżyk**. Została ona stworzona przez **Alexandra Douglasa** w ramach jego pracy doktorskiej w 1952 roku.[2] Rewolucja przyszła jednak dopiero dwadzieścia lat później, za sprawą **Ponga**. Okazał się on wielkim sukcesem i szybko wszedł do obiegu komercyjnego. Od tego czasu gry komputerowe rozwijały się w błyskawicznym tempie i przybierały coraz różniejsze formy. W dzisiejszych czasach gry możemy podzielić na niezliczoną liczbę kategorii. Jedną z nich są gry programistyczne. To specjalna kategoria gier, która łączy rozrywkę z nauką umiejętności programistycznych. W przeciwieństwie do standardowych gier, gracz nie kontroluje gry za pomocą klawiatury i myszy. Zamiast tego odpowiedzialny jest za napisanie fragmentu kodu, który będzie podejmował decyzje i wykonywał działania w ramach rozgrywki. Decyzje podejmowane są w ramach logiki zawartej w kodzie.

Aktualnie istnieją gry programistyczne na każdym poziomie zaawansowania, począwszy od prostych gier przystosowanych do nauki od podstaw, aż po zaawanso-

wane gry wymagające szczegółowej wiedzy z zakresu programowania, algorytmiki, matematyki i uczenia maszynowego. Niektóre gry przedstawione są za pomocą języka wizualnego, aby jeszcze bardziej ułatwić proces nauki i rozumienie metafor.

Gry programistyczne możemy podzielić na dwie kategorie, które znacząco różnią się z punktu widzenia gracza (programisty): jednoosobowe i wieloosobowe

1.2.1. Gry jednoosobowe (puzzle)

W grach typu puzzle gracz musi pokonać wyznaczoną planszę. Po drodze mierzy się z potworami lub przeszkodami, czasami musi zdobyć odpowiednie przedmioty lub dotrzeć do wyznaczonego celu (potencjalnie w ograniczonym czasie). W celu weryfikacji, czy cel gracza został osiągnięty, do gry dołączone są odpowiednio przygotowane testy.

Za jedne z pierwszych gier z tej kategorii uznaje się **System 15000** z roku 1984 oraz wydaną rok później grę **Hacker**[3]. W obu z nich gracz wcielał się w postać hakera i miał na celu włamać się do systemu, aby uzyskać niezbędne informacje. Wśród gier typu puzzle wyróżnić możemy również dodatkową podkategorię jaką są gry optymalizacyjne. W grach optymalizacyjnych gracz oprócz przejścia planszy musi również uzyskać w niej najlepszy możliwy wynik.

1.2.2. Gry wieloosobowe

W grach wieloosobowych toczymy pojedynki z innymi graczami. W trakcie gry symulowane są boty napisane przez graczy. Celem gracza jest zazwyczaj doprowadzenie do zniszczenia przeciwnika lub osiągnięcie lepszego wyniku. Na podstawie wygenerowanych pojedynków tworzone są listy rankingowe. Opisywana gra **Clash of Bots** jest grą dla dwóch graczy, a więc wieloosobową.

1.3. Sztuczna inteligencja w grach

Okazuje się, że koncept sztucznej inteligencji był rozważany już w latach 40. ubiegłego wieku. Zajmował się nim między innymi **Alan Turing**, który w roku 1950 stworzył **The Imitation Game**[4], dziś znaną pod nazwą **test Turinga**. Początkowo nie była ona programem komputerowym, a jedynie projektem zapisanym na papierze. Wzorowała się ona na popularnej w tamtych czasach grze polegającej na odgadywaniu płci. Przebieg gry był bardzo prosty. Sędzia rozmawiał jednocześnie z mężczyzną i kobietą, których nie widział, za pomocą kartek. Na tej podstawie miał określić płeć każdego z nich. W teście Turinga jedna z osób miała zostać zamieniona na sztuczną inteligencję. Jeśli sędzia (człowiek) nie był w stanie stwierdzić, który z rozmówców jest maszyną, uznawano, że sztuczna inteligencja zaliczyła test Turinga.

W dzisiejszych czasach sztuczna inteligencja jest zdolna do oszukania znacznej liczby sędziów.

Twórcy gier od początku dążyli do tworzenia rozwiązań, które będą jak najlepiej symulować zachowania ludzkie, tak aby gracze mieli wrażenie gry z realnym przeciwnikiem. Pierwsze algorytmy były oparte na prostych mechanizmach śledzenia gracza – tak jak na przykład odbijanie paletki w Pongu. Dziś algorytmy te są na tyle rozwinięte, że w wytworzonych mechanicznie postaciach niegrywalnych (ang. *non-player character (NPC)*) możemy dostrzec ludzkie emocje.

AI jest wykorzystywana również do tworzenia dynamicznego świata gry, co zaobserwować można między innymi w grze **Red Dead Redemption 2**. Przykładowo, NPC, będąc świadkiem przestępstwa, stara się dostać do najbliższej siedziby szeryfa, a wymiar sprawiedliwości nabiera odpowiedniej wiedzy tylko, gdy donos się powiedzie. Jeszcze bardziej imponującym przykładem może być **Shadow of Mordor/War**, gdzie system **Nemesis** potrafi uczynić z każdego podejścia do gry niepowtarzalną historię z unikalnymi, świetnie adaptującymi się na postawie interakcji z graczem (i innymi NPC-tami) *aktorami*.

Mechanizm działania sztucznej inteligencji wydawać mógłby się bardzo prosty. Maszynie zostają dostarczone pewne dane wejściowe opisujące aktualne środowisko, a algorytm powinien je odpowiednio przetworzyć i dostarczyć danych wyjściowych. Na przykład, symulując zachowanie bota, możemy powiedzieć, że jeśli zajdzie sytuacja A, bot powinien zareagować w sposób Z. Sprawa utrudnia się, kiedy czynników wpływających na zachowanie robota jest więcej. Na przykład, kiedy zajdzie sytuacja A, w środowisku B zareaguj w sposób Y. Czynniki wpływające na zachowanie maszyny w teorii mogą być dowolnie duże. Tu pojawia się główny problem twórców sztucznych inteligencji, czyli tworzenie takich algorytmów, które potrafią przetworzyć i odpowiednio priorytetować wszystkie czynniki, które mogą mieć znaczenie dla wyboru zachowania bota.

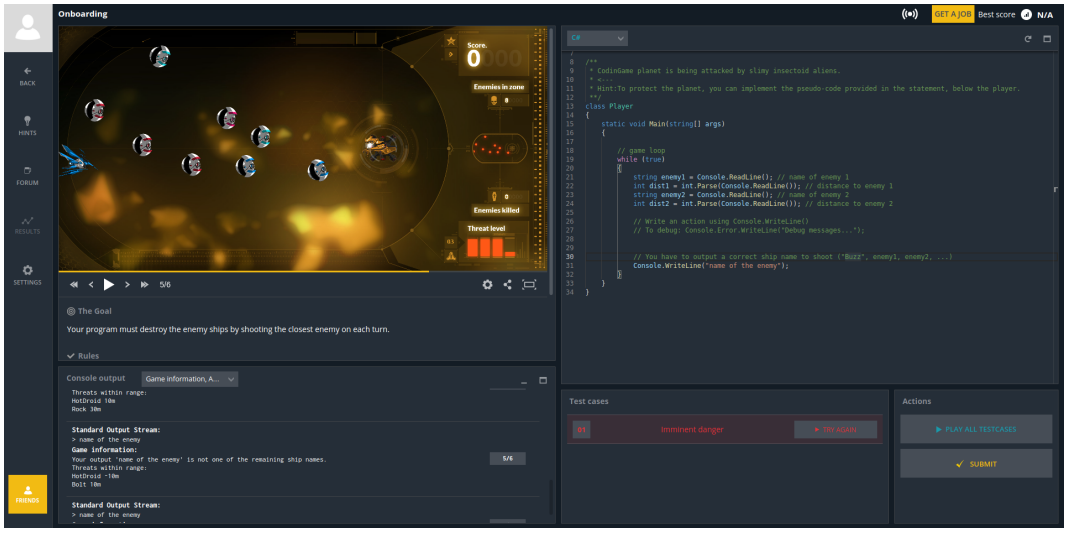
1.4. CodinGame

1.4.1. Platforma

CodinGame[1] to strona, która umożliwia rozwój umiejętności programistycznych w przyjazny sposób. Użytkownik znajdzie na niej gry o różnych stopniach trudności, pomagające opanować między innymi algorytmy sztucznej inteligencji i składnie języków programowania. Szeroki wachlarz dostępnych języków czyni stronę dostępną dla dużego grona odbiorców. Na platformie możemy zagrać zarówno w gry typu puzzle, jak i gry wieloosobowe. Zazwyczaj poczynania gracza są animowane, co ułatwia śledzenie działania algorytmu. Rozwiązanie łamigłówki lub dobre wyniki w grze wie-

[1] https://www.codingame.com

loosobowej są nagradzane punktami doświadczenia oraz osiągnięciami (ang. *achievements*), a ranking wyników wprowadza element rywalizacji.



Rysunek 1.1: Przykładowa gra na platformie CodinGame

1.4.2. Framework

Platforma CodinGame udostępnia szereg narzędzi deweloperskich umożliwiających tworzenie własnych gier [5]. Znajdziemy w nim klasy takie jak:

- GameManager – klasa dzieli się na SoloGameManager (w przypadku gier jednoosobowych) oraz MultiplayerGameManager (w przypadku gier wieloosobowych). Klasa ta zarządza przebiegiem rozgrywki, rozpoczyna i kończy poszczególne tury. Można dzięki niej ustalić ograniczenia na rozgrywkę, takie jak na przykład limit czasu odpowiedzi, zarządzać przebiegiem i zakończeniem gry. Dzięki klasie GameManager mamy też dostęp do informacji o poszczególnych graczach. Kolejną funkcją jest zarządzanie wynikiem. W grze typu puzzle odpowiada on również za przeprowadzenie testów.

- GraphicEntityModule – klasa umożliwia implementację grafiki gry. Dzięki niej możemy dodawać kształty, obrazki czy teksty, a także je animować, obracać, skalować i przesuwać. Umożliwia ona również zarządzanie animacjami w czasie trwania tury. Dzięki temu pewne animacje mogą być wyświetlane przed innymi. Automatycznie, bez dodatkowych ustawień wszystkie animacje zostaną wyświetlone na koniec tury.

- GameRunner – pozwala na uruchomieniu gry oraz jej wizualizacje w środowisku lokalnym.

1.5. Użyte technologie

Wszystkie klasy opisane we wcześniejszym podpunkcie zaimplementowane zostały w języku **Java**. Dlatego to właśnie w tym języku powinny zostać napisane gry dedykowane na tę platformę. Alternatywą są języki działające na maszynie wirtualnej Javy (czyli popularnym **JVM**ie). Takim językiem jest nowoczesny **Kotlin**, w którym została zaimplementowana gra **Clash of Bots**. Za warstwę front-endową odpowiada **JavaScript**.

Rozdział 2.

Implementacja

2.1. Opis gry

2.1.1. Plansza

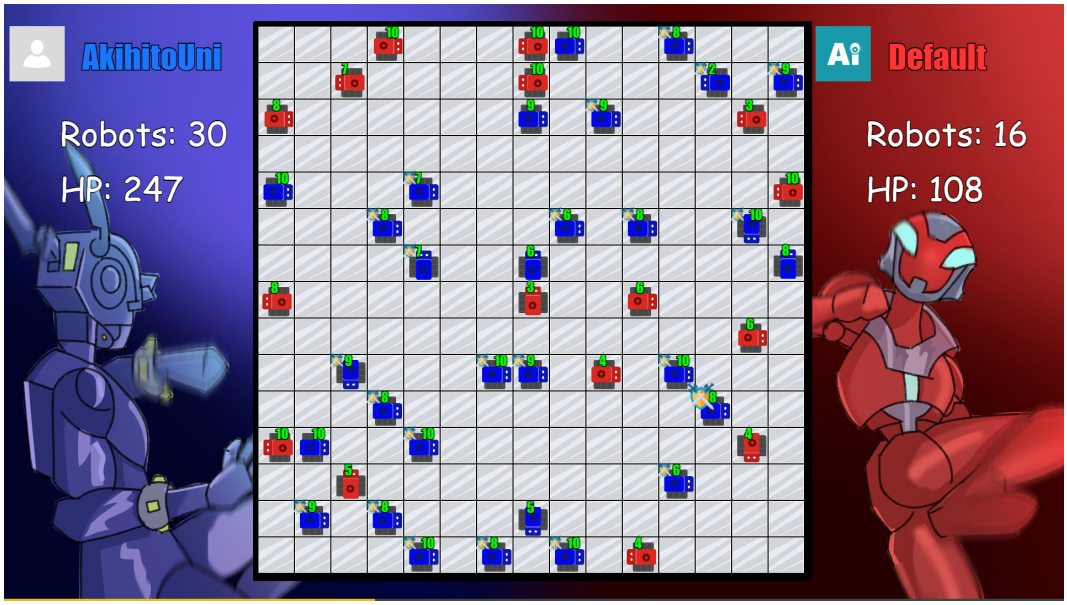
Widok gry składa się z paneli użytkowników oraz areny po której poruszają się boty. W panelach użytkownika znajdujących się po lewej i prawej stronie areny wyróżnić można:

- awatar,
- pseudonim,
- sumaryczną liczbę robotów,
- sumaryczną liczbę punktów życia wszystkich robotów posiadanych przez gracza.

Arena składa się z 225 pól (plansza o wymiarach 15×15), po których mogą poruszać się roboty. Przykładową planszę w trakcie rozgrywki można zobaczyć na rysunku 2.1.

2.1.2. Zasady

Na początku rozgrywki obaj gracze posiadają dwa roboty, rozłożone na planszy w sposób losowy, lecz symetryczny. Co *pięć* rund w grze pojawia się osiem nowych robotów – po cztery dla każdego gracza. Nowe roboty posiadają po 10 punktów życia. W każdej rundzie gracz otrzymuje dwie informacje: liczbę żywych robotów, oraz pole widzenia każdego z robotów. Robot jest w stanie dostrzec pole o rozmiarach 5x5 komórek. W pojedynczej turze każdy robot musi wykonać jedną z dostępnych akcji.



Rysunek 2.1: Przykładowa plansza podczas rozgrywki

2.1.3. Dostępne akcje

- Podniesienie gardy – robot, który w danej turze ma podniesioną gardę otrzymuje obrażenia zredukowane o 50%.
- Przemieszczenie się – robot może przemieścić się o jedno pole w pionie lub poziomie. Plansza na końcach zawija się, dlatego opuszczenie jej jest niemożliwe. Jeżeli robot spróbuje przemieścić się na pole, na którym już znajduje się inny robot, lub dwa roboty będą chciały zająć to samo pole, wtedy dojdzie do kolizji. Roboty zostaną wycofane na wcześniej zajmowane pola, a ich punkty życia zostaną zmniejszone o jeden punkt.
- Atak – możliwy jest w jednym z czterech dostępnych kierunków. Zasięg ataku to jedno pole, więc aby robot przeciwnika został trafiony, musi on znajdować się na polu bezpośrednio sąsiadującym z naszym robotem. Nie można atakować na ukos. Trafiony robot traci dwa punkty życia.
- Samozniszczenie – robot może wywołać kontrolowaną eksplozję tracąc tym samym wszystkie dostępne punkty życia. Wszystkie roboty znajdujące się na sąsiednich polach (również ukośnie) otrzymują cztery punkty obrażeń.

Wszystkie akcje przetwarzane są w podanej wyżej kolejności. Oznacza to, że robot podnoszący gardę chroniony jest przed wszystkimi atakami dostępnymi w danej turze. Garda zdezaktywuje się automatycznie na samym początku kolejnej tury. Ponieważ ruch odbywa się przed atakiem, robot może uciec przed wymierzonym w niego ciosem. Możliwe jest również, że robot, który dostał polecenie samodestrukcji, zginie zanim zdoła ją wykonać.

Rysunek 2.2: Wynik gry, w przypadku dyskwalifikacji jednego z graczy

2.1.4. Warunek końca

Gra kończy się, gdy:

- zostanie rozegranych 100 tur, lub
- jeden z graczy nie dostarczy odpowiedzi w wyznaczonym czasie, lub
- jeden z graczy dostarczy nieprawidłowe polecenie.

Jeśli pomyślnie uda się rozegrać 100 tur, wygrywa ten gracz, któremu na końcu gry pozotało więcej żywych robotów. Jeśli liczba ta jest równa, wygrywa ten, którego suma punktów życia pozostałych robotów jest wyższa. Jeśli obie te liczby są równe, odnotowany zostaje remis.

Niedostarczenie pełnej odpowiedzi w wyznaczonym czasie lub dostarczenie nieprawidłowego polecenia skutkuje dyskwalifikacją.

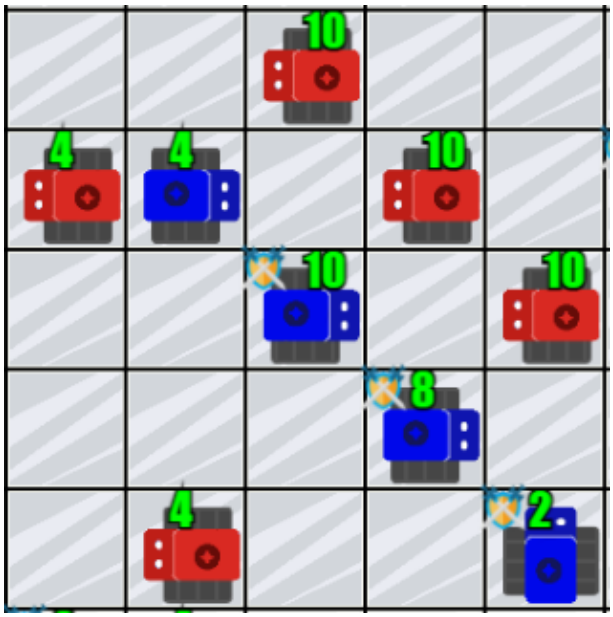
2.1.5. Protokół – dane wejściowe i wyjściowe

W pierwszej linii danych wejściowych użytkownik otrzymuje n – liczbę posiadanych robotów. W kolejnych $5n$ wierszach otrzymujemy informacje o najbliższym otoczeniu każdego z robotów. Każde kolejne 5 wierszy reprezentuje widok pojedynczego robota. Robot jest w stanie dostrzec pole o wielkości 5×5, dlatego każdy z wierszy zawiera pięć cyfr z zakresu od -10 do 10. Liczby ujemne reprezentują roboty przeciwnika, liczby dodatnie nasze roboty, zaś 0 świadczy o braku robota na

danym polu. Liczba odzwierciedla punkty życia danego robota (razy -1 w przypadku przeciwnika). Przykładowy widok dla pojedynczego robota:

```
0 0 -10 0 0
-4 4 0 -10 0
0 0 10 0 -10
0 0 0 8 0
0 -4 0 0 2
```

Obraz 2.3 przedstawia robota, dla którego widok reprezentowany byłby w powyższy sposób:



Rysunek 2.3: Sytuacja robota (znajdującego się pośrodku), dla powyższych danych wejściowych

Sędzia (podrozdział 2.2.4.) oczekuje na wyjściu n wierszy, gdzie i-ty wiersz zawiera polecenie dla i-tego robota. Polecenie jest postaci:

- `GUARD`
- `ATTACK UP`
- `ATTACK DOWN`
- `ATTACK RIGHT`
- `ATTACK LEFT`
- `MOVE UP`
- `MOVE DOWN`

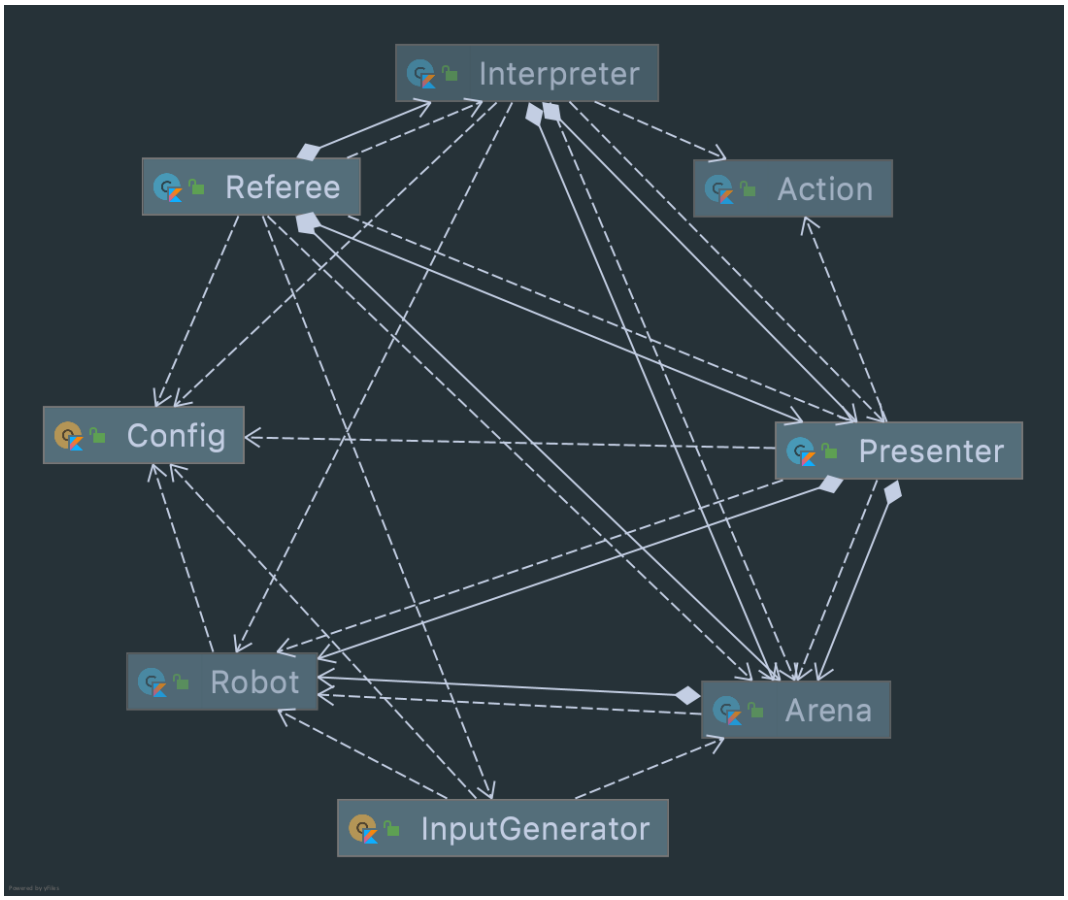
- `MOVE RIGHT`
- `MOVE LEFT`
- `SELFDESTRUCTION`

Opcjonalnie za poleceniem możemy umieścić dodatkową wiadomość, która może przydać się w celu debugowania działania naszego agenta.

Przykład: `"ATTACK UP dostrzeżono wroga na północy"`

2.2. Struktura programu

Silnik gry jest podzielony na kilka klas-modułów:



Rysunek 2.4: Diagram klas

2.2.1. Robot

Bezpośrednia reprezentacja robota. Przede wszystkim przechowuje jego aktualny stan.

2.2.2. Arena

Reprezentacja areny, na której toczą się boje. Pozwala na dodawanie, usuwanie i przesuwanie robotów oraz odpytywanie o aktualne położenie.

2.2.3. Action

ADT (Algebraic Data Type) opisujące wszystkie dostępne akcje. W tym module znajduje się również parser o sygnaturze:

```
fun tryParse(userOutput: String): Try[Action]
```

2.2.4. Referee

Odpowiada za przebieg całej rozgrywki. Ustala parametry takie jak maksymalny czas odpowiedzi czy liczba tur. W odpowiednim momencie dokłada na mapę kolejne roboty, starając się to zrobić sprawiedliwie - wybiera spawn pointy symetrycznie. W każdej turze generuje (z pomocą InputGeneratora) dane wejściowe oraz wysyła je do graczy a odpowiedź zwrotną przekazuje do Interpretera. Jeśli któryś z graczy dostarczył nieprawidłową komendę lub nie dostarczył odpowiedzi w odpowiednim czasie, to zostaje ten gracz zdyskwalifikowany. Na końcu rozgrywki sumuje punkty, wyznaczając tym zwycięzcę.

2.2.5. Interpreter

W tej klasie zaimplementowane są główne mechaniki gry. Odpowiada za faktyczne wykonanie wszystkich akcji, zgodnie z regułami rozgrywki. Dodatkowo informuje warstwę prezentacji o zmianach w stanie gry.

2.2.6. Presenter

Odpowiada za warstwę wizualną gry. Rysuje plansze oraz animuje przebieg rozgrywki.

2.2.7. InputGenerator

Dla danego robota oraz aktualnego stanu areny, generuje dane wejściowe dla gracza.

Rozdział 3.

Boty

3.1. Bot autorski

Bot autorski został stworzony z myślą o nadanie mu charakteru domyślnego przeciwnika dla graczy. To na nim swoje boty mogliby testować gracze, jeszcze przed wysłaniem ich na arenę. Z tego powodu, bot ten nie mógł być przesadnie złożony, tak aby nie zniechęcać początkujących graczy.

Logika bota opiera się na czterech warunkach, które są sprawdzane w określonej kolejności dla każdego robota. Jeśli dany warunek zostanie spełniony, zostaje podjęta akcja danego robota, a kolejne warunki nie są sprawdzane.
Na początku sprawdzone zostaje czy robotowi opłaca się dokonać samozniszczenia. Warunek ten zostanie spełniony, tylko wtedy kiedy liczba robotów przeciwnika, którą może potencjalnie zniszczyć jest co najmniej o dwa wyższa niż liczba zniszczonych robotów z własnej drużyny.

Następnie sprawdzone zostaje, czy robot ma wrogie roboty w zasięgu swojego ataku. Ataki są rozpatrywane w kolejności góra, prawo, dół i lewo. Kolejność ta jest ważna, ponieważ, jeśli jeśli atakujemy w innym kierunku niż do góry, bot sprawdza również, czy atakowana jednostka nie zostanie wcześniej zniszczona przez innego robota. Na przykład jeśli bezpośrednio pod nami znajduje się wrogi robot z dwoma punktami życia, a pod nim inny robot z naszej armii, to zostanie on w pierwszej kolejności zaatakowany przez innego robota, a tym samym zniszczony. W takiej sytuacji nie opłaca się nam go atakować.

Jako kolejną akcję bot rozważa możliwość przemieszczenia się. Należy tu zauważyć, że ponieważ akcja ta rozgrywa się po sprawdzeniu możliwości ataku, oznacza to, że dojdzie do jej wykonania tylko pod warunkiem, że na żadnym z sąsiadujących pól nie stoi wrogi robot. Robot przemieści się tylko wtedy, kiedy w wyniku tej akcji znajdzie się w bezpośrednim sąsiedztwie z wrogim robotem, obok którego znajduje się już przynajmniej jeden z naszych robotów. Dzięki temu istnieje szansa otoczenia wroga.

Jeżeli nie zajdą warunki odpowiednie dla żadnej z tych akcji bot ponownie sprawdzi czy warto dokonać samodestrukcji. Tym razem, sprawdzi jednak czy suma punktów życia zabranych przeciwnikowi, będzie większa niż utracona przez własną armię. Do tej akcji nie dojdzie jednak, jeśli robot posiada więcej niż cztery punkty życia. Jeżeli żaden z warunków nie zostanie spełniony, bot w ramach akcji podniesie gardę.

3.2. Analiza botów graczy

Jednym z wymagań jakie platforma CodinGame stawia przed twórcami gier, jest konieczność stworzenia takiej gry, aby najprostszego bota można było napisać w jednej linijce. Ma to zachęcić początkujących graczy do wzięcia udziału w rozgrywce. W grze Clash of Bots, również możemy stworzyć boty, które zawsze podejmują tę samą akcję. Obserwując arenę, możemy dostrzec roboty, które zostały skonstruowane właśnie na takiej zasadzie. Roboty te, niezależnie od sytuacji zawsze podnoszą gardę lub tylko atakują w jednym kierunku.

Nieco bardziej rozwinięte boty, analizują najbliższe komórki robota i wybierają kierunek ataku, w którym powinien znaleźć się przeciwnik. Wyżej w rankingu zauważymy boty, które działają na podobnej zasadzie, jednak w przypadku nie odnalezienia przeciwnika poruszają się w jednym kierunku, w celu zlokalizowania wroga. Kolejnym krokiem rozwoju jest analizowanie stopnia zagrożenia sytuacji, w której znajduje się robot i w razie potrzeby podniesienie gardy lub dokonanie samodestrukcji. W rankingu możemy znaleźć również graczy, których boty w pełni analizują sytuację na planszy i starają się dobrać najlepsze możliwe rozwiązanie za pomocą bardziej zaawansowanych algorytmów.

Rozdział 4.

Podsumowanie

W pracy zostały opisane szczegóły implementacji gry Clash of Bots. Możemy w niej znaleźć podstawowe założenia, które kierowały autorami pracy, zasady gry oraz opis użytych technologii. Gra została opublikowana na platformie CodinGame i umożliwia graczom rozwój umiejętności programistycznych. Do implementacji gry został użyty język Kotlin.

Jak wynika z analizy botów pierwszych graczy, gra jest przyjazna zarówno dla początkujących graczy, jak i tych posiadających większe doświadczenie z programowaniem. W czołówce rankingu znaleźć możemy graczy, którzy do stworzenia swoich botów wykorzystali bardziej zaawansowane algorytmy.

W pracy został opisany również bot autorski, który ma zostać opublikowany jako domyślny przeciwnik. Gracze będą mogli mierzyć się z nim, zanim spróbują swoich sił przeciwko innym użytkownikom. W przyszłości, w ramach rozwoju programu, można dalej udoskonalać bota, tak aby mógł się on stać bossem w grze. Boss to przeciwnik, którego pokonanie umożliwia graczom zdobywanie nowych osiągnięć w platformie.

Bibliografia

[1] CodinGame https://www.codingame.com/

[2] Steffen P. Walz *Toward a Ludic Architecture: The Space of Play and Games* ETC Press, 2010

[3] *Programing Game* https://en.wikipedia.org/wiki/Programming_game

[4] Jan Argasiński *Sztuczna Inteligencja w grach wideo* https://ruj.uj.edu.pl/xmlui/bitstream/handle/item/55252/argasinski_sztuczna_inteligencja_w_grach_wideo_2012.pdf?sequence=1&isAllowed=y

[5] *CodinDame SDK documentation* https://www.codingame.com/playgrounds/25775/codingame-sdk-documentation/introduction