Raport

Marcin Woźniak
Filip Izydorczyk
Hubert Wrzesiński
Przemysław Fierek

Grupa 5

12 czerwca 2019

Repozytorium: `https://github.com/linux923344/autonomiczny_saper/`

Spis klas użytych w projekcie:

1. Board

   (i) Board - jako argumenty przyjmuje X i Y, które sa rozmiarem renderowanego okna. Klasa odpowiada za renderowanie mapy i zarzadzanie obiektami na niej.

   (ii) Direction - enum, który daje nam informacje, w którym kierunku ma sie poruszać postać.

   (iii) DirectionCalculator - obliczanie następnego punktu na podstawie podanych wcześniej x i y.

   (iv) EquipmentGui - klasa odpwoiedzialna za grafikę ekwipunku

   (v) EquipmentGuiControl - klasa odpowiedzlana za zbieranie narzędzi i umieszczanie ich w ekwipunku.

   (vi) GameStarter - odpowiada za wystartowanie całego programu, po wybraniu odpowiednich opcji z menu.

   (vii) MapReader - argumentem przy jej tworzeniu jest plansza czyli obiekt typu Board, na której MapReader bedzie tworzy l obiekty które sa przechowywane w zewnetrznym pliku.

   (viii) Point - klasa punktu

   (ix) WalkingType - typ chodzenia (Za pomocą uczenia maszynowego, lub algorytmów chodzenia)

2. MapObjects

(i) Bombs:

- BombRed
- BombBlue
- BombYellow

} Obiekty odpowiedzialne za renderowanie oraz logikę bomby

(ii) Saper - przechowuje informacje potrzebne do wyrenderowania Sapera, podnoszenie rzeczy do ekwipunku oraz jakie zadania musi wykonać.

(iii) Stone - klasa, która jest odpowiedzialna za informacje potrzebne do wyrenderowania kamienia.

(iv) Tool - klasa, która jest odpowiedzialna za informacje potrzebne do wyrenderowania na- rzedzia.

(v) Water - klasa, która jest odpowiedzialna za informacje potrzebne do wyrenderowania wody.

3. MenuWindow

(i) MenuWindow - klasa odpowiedzialna za wygenerowanie menu, dzięki któremu możemy wybrać mapę, po której chcemy się poruszać, oraz typ algorytmu, za pomocą któ©ego chcemy przejść daną mapę.

(ii) AlgorythmType - enum, z pomocą którego wybieramy algorytmy w MenuWindow.

4. Path Finder

(i) BestFirstSearch

- Vertex - tworzy wierzchołki grafu
- GraphCreatorBestFS - tworzy graf BestFirstSearch
- GraphBFS - przejście po grafie BestFirstSearch

(ii) BreathFirstSearch

- Vertex - tworzy wierzchołki grafu
- GraphCreatorBFS - tworzy graf BFS
- GraphBFS - przejście po grafie BFS

(iii) DepthFirstSearch

- Vertex - tworzy wierzchołki grafu
- GraphCreator - tworzy graf DFS
- Graph - przejście po grafie DFS

5. PathFinder- wywołuje poszczególne algorytmy na planszy

6. VowpalVabbit

(i) DataCreator - klasa która tworzy dane uczące dla Vowpal Wabbit

(ii) CellState - stan komórki

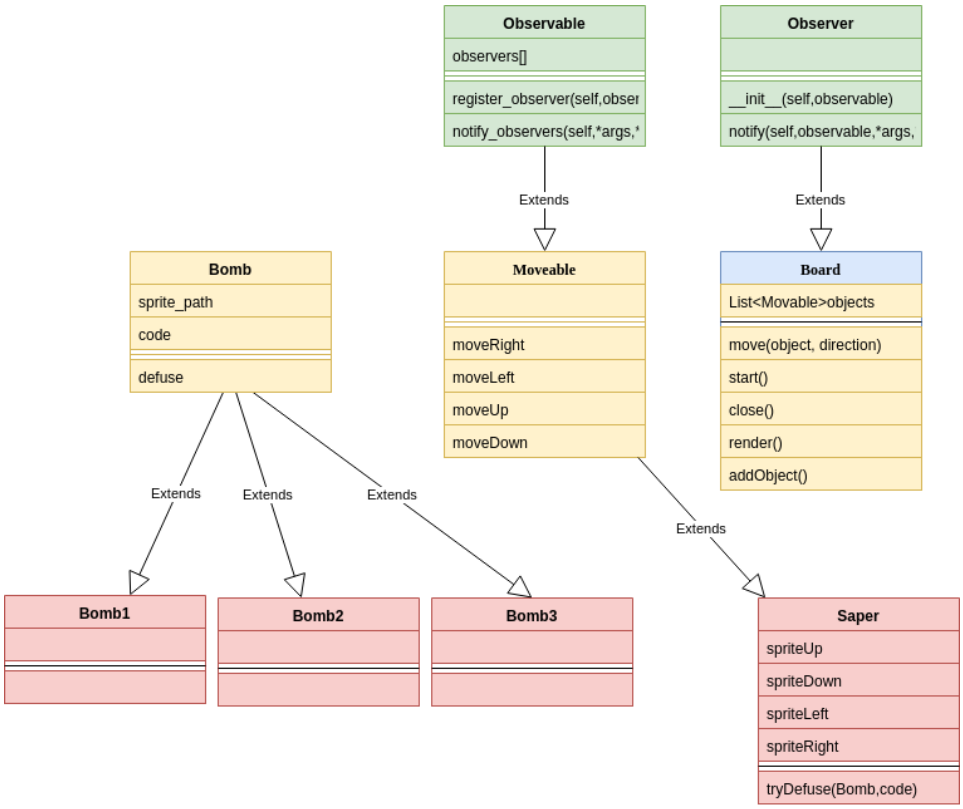
(iii) VowpalPredicter - przewidywanie kolejnych kroków za pomocą Vowpala

(iv) TreeCreator - tworzy model drzewa decyzyjnego

(v) DecisionTreePredicter - zwraca predykcje na podstawie wcześniej utworzonego modelu

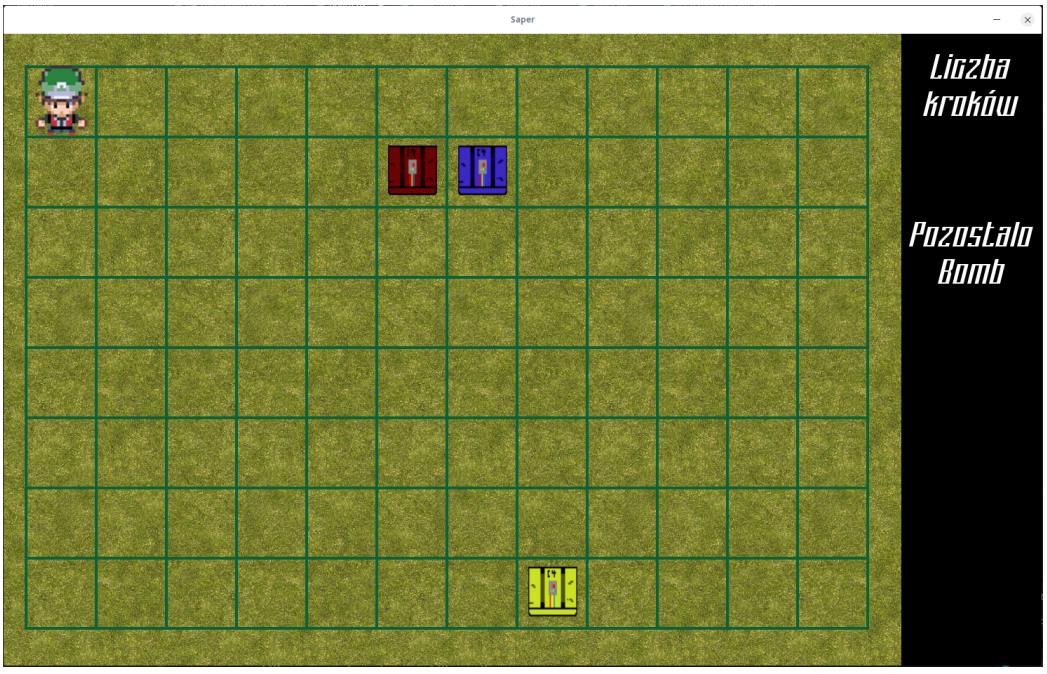
27.03.2019 r.

1. Zaprojektowanie diagramu klas.



2. Na podstawie diagramu, została stworzona klasa główna oraz jej podklasy.

3. Został stworzony szablon planszy razem z umieszczonymi na niej bombami.



Na załączonym zrzucie ekranu widoczny jest saper czyli nasz Agent, oraz trzy kolorowe obiekty (bomby), które nasz agent ma za zadanie je rozbroić.

10.04.2019 r.

W tym dniu została dokończona reprezentacja wiedzy w naszym projekcie. Przedstawimy ją teraz:

1. Posiadamy trzy rodzaje bomb (czerwona, żółta, niebieska).

2. Aby rozbroić bombę czewoną musimy posiadać narzędzie dodatkowo zmieścić się w wyznaczonym czasie. Jednostką czasu w naszym świecie jest jeden krok.

3. Aby rozbroić pozostałe bomby Saper musi tylko do nich podejść, oczywiście jak nakrótszym czasie.

4. Saper będzie wyznaczał drogę za pomocą algorytmu przeszukiwania grafu.

15.04.2019 r.

W tym dniu został zaimplementowany wyznaczanie ścieżki za pomocą algorytmu przeszukiwania DFS oraz kilka klas.

17.04.2019 r.

Spisanie raportu i wypisanie wszystkich klas znajdujących się w projekcie.

30.04.2019 r.

Dodanie ekwipunku i ustalenie zasad działania bomb. Bomby będą miały określony czas, a narzędzia będą potrzebne do rozbrojenia ich.

6.05.2019 r.

Implementacja algorytmu BFS.

14.05.2019 r.

Implementacja algorytmu Best-First Search.

28.05.2019 r.

Dodanie klasy tworzącej dane uczące. Agent przechodzi graf, i z każdym krokiem zapisuje stan pól znajdujących się dookoła niego w kwadracie 7x7, do pliku.

29.05.2019 r.

Ulepszenie algorytmu Best-First Search. Teraz nasz agent przechodzi graf zbierając po drodze wszystkie narzędzia znajdujące się na mapie

01.06.2019 r.

Wykorzystanie zapisanych danych uczących przy Vowpal Wabbicie, a także dodanie licznika do bomb.

02.06.2019 r.

Stworzenie większej ilości map do uczenia, a co za tym idzie, stworzenie większej ilości danych uczących.

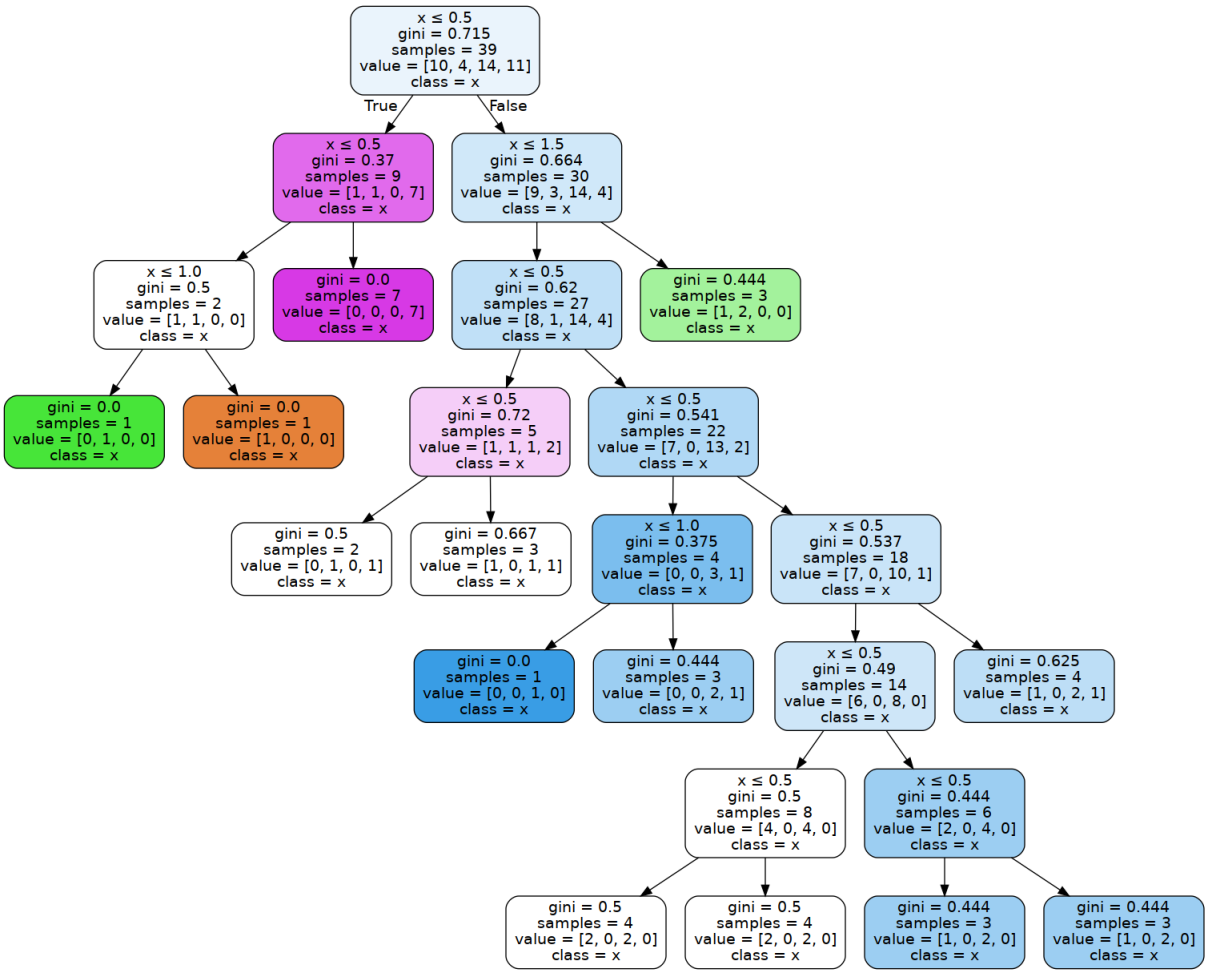
03.06.2019 r.

Agent jest nauczony, ale nie działa perfekcyjnie.

Opis zastosowania drzew decyzyjnych

Podczas uruchamiania algorytmu przechodzenia mapy, ładujemy z pliku model drzewa decyzyjnego i dopasowujemy mu dla obszaru 3 kratek w każdą stronę, liczbę będącą odpowiednikiem kierunku.

Następnie nadpisuje wcześniejszy model nowym, przy uruchomieniu programu z drzewem decyzyjnym, wczytuje model z pliku, program wysyła do modelu obszar na 3 kratki w każdą stronę, a drzewo zwraca predykcję w postaci liczby, która odpowiada danemu kierunkowi.



Opis zastosowania Vowpal Wabbit

Uruchamiamy algorytm przeszukiwania Best-First Search, który w tle tworzy dane uczące potrzebne do Vowpal Wabbit. Dane uczące pokazują nam stan każdej kratki w obrbie naszego

agenta. Przestrzeń wokół agenta, to kwadrat o promieniu 3.

Nastepnie gdy odpalamy Vowpala wysylamy obecny stan na przestrzeni 3 kratek w każdą ze stron. On nam zwraca liczbę, na podstawie której wybieramy kierunek.

0 - lewo
1 - gora
2 - prawo
3 - doł