Dokumentacja MineBot

Jakub Pietrzko

Spis treści

1. Wstęp

Sztuczna inteligencja znajduje niezwykle szerokie zastosowanie w grach komputerowych, w tym w sterowaniu postaciami niezależnymi (NPC), proceduralnym generowaniu treści, realistycznych animacjach, optymalizacji procesów testowania, interakcji z graczami w czasie rzeczywistym oraz poprawie jakości grafiki dzięki algorytmom uczenia maszynowego.

Mój projekt koncentruje się na wykorzystaniu sztucznej inteligencji do stworzenia wirtualnych pomocników, z którymi gracze mogą komunikować się w naturalnym języku dzięki zaawansowanym modelom językowym (LLM). Tradycyjne podejścia, polegające na wybieraniu z ograniczonych opcji dialogowych, ograniczają swobodę interakcji, podczas gdy moje rozwiązanie pozwala graczowi jasno wyrażać swoje potrzeby, a bot reaguje w sposób dynamiczny i elastyczny. Projekt realizowany w środowisku gry Minecraft umożliwi stworzenie inteligentnego bota, który będzie wykonywał zadania takie jak eksploracja, budowanie, walka czy tworzenie przedmiotów, korzystając z poleceń wyrażonych w języku naturalnym.

Bot będzie wyposażony w zdolność samouczenia się, tworząc i korzystając z poradników, które sam opracuje na podstawie doświadczeń w grze. Dzięki temu jego działania będą coraz bardziej efektywne, a podejście adaptacyjne pozwoli mu dostosowywać się do zmieniających się warunków. Co więcej, integracja z biblioteką Selenium umożliwi botowi dostęp do zasobów internetowych, co otwiera nowe możliwości, takie jak zdobywanie dodatkowej wiedzy czy podejmowanie bardziej świadomych decyzji w oparciu o aktualne dane.

Projekt ten nie tylko rozwija potencjał gier jako środowiska do testowania zaawansowanych modeli SI, ale również przypomina koncepcje sterowania fizycznymi robotami w rzeczywistości. Gry takie jak Minecraft oferują uproszczony, lecz złożony świat, w którym można sprawdzić zdolności modeli do rozwiązywania problemów, uczenia się oraz generowania wieloetapowych strategii.

Sterowanie botem przy użyciu dużych modeli językowych ukazuje możliwości tworzenia bardziej „ludzkich” zachowań w środowiskach cyfrowych. Integracja takich technologii w grach komputerowych to krok w stronę głębszego zrozumienia ich potencjału oraz ograniczeń. W przyszłości podobne rozwiązania mogą znaleźć zastosowanie w rzeczywistych kontekstach, takich jak sterowanie autonomicznymi robotami, zarządzanie inteligentnymi systemami miejskimi czy rozwój interaktywnych asystentów osobistych.

2. Środowisko i mechaniki gry Minecraft

Minecraft to gra typu sandbox, w której światy są generowane proceduralnie, co sprawia, że każdy nowy świat jest wyjątkowy i niepowtarzalny. Podstawową jednostką budowy świata stanowią sześcienne **bloki**, reprezentujące różnorodne materiały, takie jak drewno, kamień, ziemia czy minerały. Światy w grze dzielą się na różnorodne **biomy**, takie jak lasy, pustynie, góry czy oceany, z których każdy charakteryzuje się unikalnymi cechami środowiskowymi. Dodatkowo struktury, takie jak wioski, świątynie czy lochy, są rozmieszczane w wybranych biomach zgodnie z określonymi zasadami generacji.

Grafika. 2.1. Środowisko Minecraft w którym bot uczył się różnych funckjonalności, zrzut ekranu, źródło własne

2.1. Podstawowe mechaniki gry

Jednym z kluczowych elementów gry jest zbieranie zasobów i tworzenie z nich przedmiotów. Proces ten obejmuje:

- **Zbieranie surowców**: Drewno, kamień, węgiel, żelazo i inne materiały stanowią podstawę przetrwania i rozwoju.
- **Crafting**: Wykorzystanie stołu warsztatowego (*crafting table*) umożliwia tworzenie narzędzi, broni, zbroi oraz innych przedmiotów niezbędnych do rozgrywki.

Zebrane zasoby pozwalają na budowę schronień, farm, a także skomplikowanych systemów logicznych z użyciem **Redstone**, czy tworzenie pięknych i złożonych budynków, takich jak te przedstawione na ilustracji 7.1. W poszukiwaniu rzadkich surowców gracze muszą często odkrywać nowe biomy, jaskinie oraz inne wymiary, takie jak Nether czy End.

2.2. Survival i walka

W grze występuje cykl dnia i nocy oraz system zmiennej pogody, które wpływają na przebieg rozgrywki. W nocy, a także podczas burz, pojawiają się wrogie moby, takie jak zombie i szkielety, które stanowią zagrożenie zarówno dla gracza, jak i dla bota.

Tryb przetrwania (*survival*) wymaga zarządzania zdrowiem oraz głodem, co zmusza gracza do regularnego zdobywania pożywienia. Dodatkowym wyzwaniem jest walka z wrogimi mobami, takimi jak creepery, szkielety czy pająki, co wymaga strategicznego podejścia oraz skutecznego wykorzystania dostępnych narzędzi i broni.

2.3. Interakcja z AI w grze

Sztuczna inteligencja w Minecraftcie obejmuje różnorodne moby, które można podzielić na trzy główne kategorie:

- **Przyjazne moby**: Krowy, świnie i kurczaki, które dostarczają surowców, takich jak jedzenie czy skóra.
- **Neutralne moby**: Endermany, które pozostają niegroźne, dopóki gracz ich nie sprowokuje.
- **Wrogie moby**: Zombie, szkielety i creepery, które aktywnie atakują gracza, tworząc wyzwania podczas eksploracji.

Wszystkie te mechaniki stanowią kluczowe aspekty, które muszą zostać zaimplementowane w projekcie bota, aby mógł on efektywnie funkcjonować w świecie gry.

3. Transformery i Duże Modele Językowe

W ostatnich latach transformery stały się fundamentem nowoczesnych modeli językowych, umożliwiając znaczący postęp w dziedzinie przetwarzania języka naturalnego (NLP). W tym rozdziale omówię architekturę transformera oraz jej zastosowanie w dużych modelach językowych (LLM).

3.1. Architektura Transformera

Transformery zostały wprowadzone w 2017 roku w pracy "Attention Is All You Need"[1], rewolucjonizując podejście do zadań sekwencyjnych w przetwarzaniu języka naturalnego. Tradycyjne modele, takie jak rekurencyjne sieci neuronowe (RNN), przetwarzały dane sekwencyjnie, co ograniczało ich zdolność do uchwycenia długoterminowych zależności. Transformery eliminują te ograniczenia dzięki mechanizmowi atencji, który pozwala modelowi ocenić zależności między wszystkimi elementami sekwencji jednocześnie.

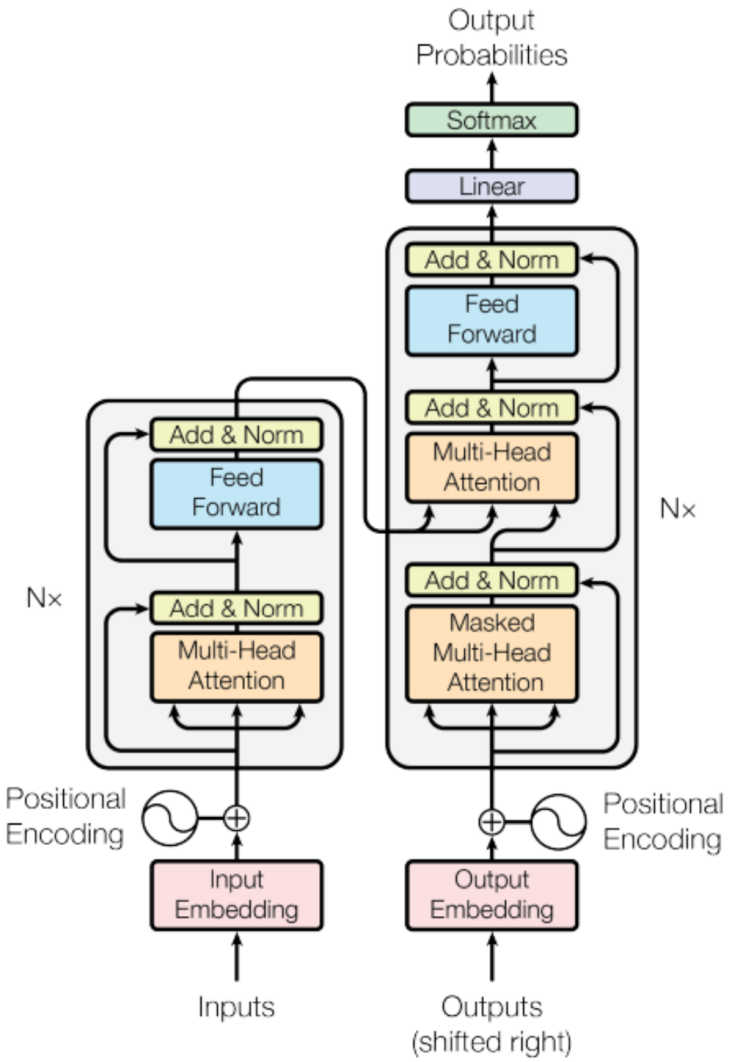
3.2. Mechanizm atencji

Mechanizm atencji (ang. attention mechanism) w sieciach neuronowych umożliwia modelom skupienie się na istotnych częściach danych wejściowych podczas przetwarzania informacji. W kontekście przetwarzania języka naturalnego, mechanizm ten pozwala modelowi ocenić, które słowa w zdaniu są najważniejsze dla zrozumienia jego znaczenia.

Jak to działa?

Dla każdego słowa w zdaniu mechanizm atencji oblicza trzy wektory:

- Query (zapytanie): reprezentuje bieżące słowo, dla którego określamy istotność innych słów.
- Key (klucz): reprezentuje każde słowo w zdaniu i służy do porównania z zapytaniem.
- Value (wartość): niesie informację o słowie i jest wykorzystywana do tworzenia nowej reprezentacji zdania.



Grafika. 3.1. Architektura sieci transformer, źródło: artykuł "Attention Is All You Need"[1]

Następnie dla każdego słowa obliczana jest miara podobieństwa między jego zapytaniem a kluczami wszystkich słów w zdaniu. Miary te są normalizowane, aby uzyskać tzw. wektory uwagi (attention vectors), które wskazują, na które słowa model powinien zwrócić większą uwagę. Ostatecznie wartości są ważone tymi współczynnikami uwagi i sumowane, tworząc nową reprezentację każdego słowa w kontekście całego zdania.

Dzięki temu mechanizmowi model jest w stanie efektywnie wychwytywać zależności między słowami, nawet jeśli są one od siebie oddalone w sekwencji, co znacząco poprawia jego zdolność do rozumienia i generowania języka naturalnego.

3.3. Struktura Kodera i Dekodera

Transformer składa się z dwóch głównych komponentów:

– Koder: Składa się z wielu warstw zawierających mechanizm atencji oraz sieci neuronowe typu feed-forward. Koder przetwarza sekwencję wejściową, generując jej reprezentację wewnętrzną.

– Dekoder: Również zbudowany z wielu warstw, dekoder wykorzystuje mechanizm atencji oraz uwagi względem wyjścia kodera, aby generować sekwencję wyjściową.

Wyjście dekodera jest przekazywane na drugie wejscie dekodera, dzięki czemu utrzymjemy informacje o kontekscie tego co juz model powiedział.

3.4. Duże Modele Językowe (LLM)

Duże modele językowe, takie jak GPT-3 czy GPT-4, to zaawansowane systemy AI zdolne do generowania tekstu przypominającego ludzkie wypowiedzi. Są one trenowane na ogromnych zbiorach danych tekstowych, co pozwala im na wykonywanie różnorodnych zadań, takich jak tłumaczenie, podsumowywanie czy odpowiadanie na pytania czy automatyzajca zadań do tej pory przezoanczonych tylko dla ludzi jak gra w Minecrafta.

3.5. Trenowanie LLM

Proces trenowania LLM polega na analizie ogromnych korpusów tekstowych w celu nauczenia modelu zależności i wzorców językowych. Dzięki zastosowaniu architektury transformera, modele te są w stanie efektywnie przetwarzać i generować tekst, uwzględniając kontekst całej sekwencji, a nie tylko jej fragmentów.

3.6. Zastosowanie LLM w Sterowaniu Robotem w Minecraft

W kontekście tworzenia robota w grze Minecraft sterowanego przez LLM, model językowy może być wykorzystany do interpretacji poleceń użytkownika i generowania odpowiednich akcji w grze. Dzięki zdolnościom do rozumienia i generowania języka naturalnego, LLM mogą przekształcać instrukcje tekstowe w konkretne działania, umożliwiając intuicyjną interakcję z robotem.

4. Podobne prace

W ostatnich latach rozwój zaawansowanych modeli językowych (LLM, ang. Large Language Models) przyczynił się do powstania projektów mających na celu umożliwienie modelom gry w Minecrafta. Poniżej omówiono dwa projekty, które są zbliżone do mojego i stanowiły inspirację podczas jego tworzenia.

4.1. Projekt Voyager

Voyager [2] to jeden z pierwszych autonomicznych agentów sterowanych przez LLM, zaprojektowanych do działania w otwartym środowisku Minecrafta. Celem projektu było stworzenie bota, który potrafi samodzielnie uczyć się nowych umiejętności bez ingerencji człowieka. Funkcjonalność ta zainspirowała mnie do opracowania systemu generowania poradników po zakończeniu realizacji określonego zadania. Do kluczowych cech Voyagera należą:

- **Zarządzanie wiedzą:** Voyager zdobywał nowe umiejętności podczas rozgrywki, wykorzystując zdobytą wiedzę do rozwiązywania coraz bardziej złożonych problemów.
- **Adaptacja do środowiska:** Agent potrafił dostosować swoje zachowanie do zmieniających się warunków gry, co umożliwiało bardziej elastyczne działanie.
- **Ograniczony zestaw umiejętności podstawowych:** Voyager dysponował 18 umiejętnościami podstawowymi, takimi jak kopanie, wytwarzanie przedmiotów czy eksploracja, które stanowiły fundament jego działania.

4.2. Projekt ODYSSEY

Projekt ODYSSEY[3] rozwija ideę autonomicznych agentów, oferując bardziej zaawansowaną architekturę i znacznie większy zakres umiejętności. Agent ten jest sterowany przez model LLaMA, który został dodatkowo dostrojony na dużym zbiorze danych pochodzących z Minecraft Wiki. Kluczowe cechy projektu ODYSSEY to:

- **Rozbudowana biblioteka umiejętności:** Agent dysponuje 40 umiejętnościami podstawowymi (np. kopanie, poruszanie się) oraz 183 umiejętnościami złożonymi (np. stworzenie kilofa z żelaza).
- **Architektura wieloskładnikowa:**
  - *Planner (planista):* Dzieli główny cel na logicznie uporządkowane podcele.
  - *Actor (aktor):* Wykonuje podcele, wykorzystując odpowiednie umiejętności agenta.
  - *Critic (krytyk):* Weryfikuje wyniki i wprowadza ewentualne poprawki w strategii działania.
- **Wykorzystanie zaawansowanego modelu LLaMA-3:** Model został specjalnie dostrojony do gry w Minecrafta za pomocą fine-tuningu na zbiorze danych z Minecraft Wiki, zawierającym 390 tysięcy wpisów.
- **Nowe benchmarki:** Projekt ODYSSEY wprowadził metody oceny agentów, uwzględniające m.in. zdolność do długoterminowego planowania, dynamicznego reagowania na zmiany w środowisku oraz efektywnej eksploracji świata gry.

Oba projekty stanowią ważny krok w rozwoju autonomicznych agentów działających w złożonych środowiskach. Ich osiągnięcia oraz zastosowane rozwiązania dostarczają cennych wskazówek i inspiracji, które można wykorzystać w dalszych badaniach i rozwijaniu własnych systemów opartych na modelach językowych.

5. Sterowanie botem w grze Minecraft za pomocą modeli LLM

W niniejszym rozdziale przedstawiam sposób, w jaki zaawansowane modele językowe (LLM) mogą być wykorzystywane do sterowania botem w grze Minecraft. System opiera się na dwóch modelach: LLaMA 3 70B oraz ChatGPT-4, które pełnią różne role w realizacji zadań, dostosowując swoje działania do wymagań sytuacji.

5.1. Podstawowe komendy i framework Mineflayer

Bot wykorzystywany w projekcie korzysta z frameworka Mineflayer, który umożliwia sterowanie botem w grze Minecraft za pomocą języka TypeScript. Jest on dostępny na stronie *https://github.com/PrismarineJS/mineflayer* Dzięki temu frameworkowi można definiować różnorodne komendy odpowiedzialne za działania bota, takie jak kopanie, wytwarzanie przedmiotów, eksploracja, czy walka. Przykładowe komendy mogą obejmować:

- poruszanie się do wskazanych współrzędnych,
- interakcję z blokami (np. kopanie lub stawianie bloków),
- korzystanie z ekwipunku bota.

Dodatkowo, bot korzysta z biblioteki mcdata, która dostarcza szczegółowych informacji o świecie gry, takich jak identyfikatory bloków, typy przedmiotów oraz ich właściwości. Informacje te są niezbędne do podejmowania przez bota decyzji opartych na kontekście.

5.2. Wykorzystanie modeli LLM

Do zarządzania działaniami bota i realizacji zadań zastosowano dwa modele językowe:

5.2.1. Model LLaMA 3 70B

Model LLaMA 3 70B pełni funkcję szybkiego mechanizmu decyzyjnego. Jest używany do wykonywania prostych zadań, takich jak:

– interpretacja podstawowych komend,

– sprawdzanie w bibliotece poradników, czy dane zadanie było już realizowane w przeszłości,

– podejmowanie decyzji w czasie rzeczywistym, gdy problem nie wymaga zaawansowanego myślenia.

5.2.2. Model ChatGPT-4

Model ChatGPT-4, obsługiwany dzięki bibliotece Selenium, umożliwia przeszukiwanie internetu w celu zdobycia dodatkowej wiedzy na temat gry Minecraft. Jest wykorzystywany do:

– rozwiązywania bardziej złożonych problemów wymagających dostępu do zewnętrznych źródeł informacji,

– generowania szczegółowych poradników dla trudniejszych zadań (miejsce na przykład poradnika zostawiono poniżej).

Miejsce na przykładowy wygenerowany poradnik przez ChatGPT-4.

Choć ChatGPT-4 jest bardziej zaawansowany i wszechstronny, jego działanie jest wolniejsze w porównaniu do LLaMA 3 70B. Dlatego jest używany głównie w trybie "myślenia", kiedy problem wymaga bardziej szczegółowego rozważenia. Model LLaMA 3 może wywołać ten tryb, jeśli uzna, że zadanie jest zbyt trudne do realizacji samodzielnie.

5.3. Mechanizm pamięci konwersacji

Model LLaMA 3 70B nie posiada wbudowanej pamięci własnej, dlatego zaimplementowano mechanizm pamięci konwersacji w postaci klasy `MemoryManager`. Mechanizm ten umożliwia przechowywanie ostatnich interakcji między botem a systemem. Poniżej znajduje się kod implementacji:

```
export class MemoryManager {
    private static instance: MemoryManager;
    private memory: { sender: string; question: string; answer: string
    private maxMemorySize = 4;

    private constructor() {}

    public static getInstance(): MemoryManager {
        if (!MemoryManager.instance) {
            MemoryManager.instance = new MemoryManager();
        }
        return MemoryManager.instance;
    }

    public addToMemory(sender: string, question: string, answer: string
        this.memory.push({ sender, question, answer });
        if (this.memory.length > this.maxMemorySize) {
            this.memory.shift();
        }
    }

    public getMemory() {
        return this.memory;
    }
}
export default MemoryManager;
```

Klasa `MemoryManager` została zaimplementowana jako Singleton. Oznacza to, że w systemie istnieje tylko jedna instancja tej klasy, co zapewnia spójność przechowywanej pamięci. Mechanizm ten pozwala na efektywne zarządzanie wiedzą krótkoterminową bota, umożliwiając mu podejmowanie bardziej kontekstowych decyzji.W implementacji klasy MemoryManager mechanizm przechowywania danych został zrealizowany jako kolejka FIFO (First In, First Out). Oznacza to, że elementy są przechowywane w określonej kolejności – najstarszy element, czyli ten, który znajduje się w pamięci najdłużej, jest usuwany w momencie, gdy liczba zapisanych elementów przekroczy maksymalny rozmiar pamięci (maxMemorySize).

Każda nowa interakcja (zawierająca informacje o nadawcy, pytaniu i odpowiedzi) jest dodawana na końcu tablicy reprezentującej pamięć. Gdy pamięć osiąga swoją maksymalną pojemność, mechanizm automatycznie usuwa najstarszy element, znajdujący się na początku tablicy,

korzystając z metody shift(). Dzięki temu kolejka zawsze zawiera najnowsze dane w określonym limicie pamięci, zapewniając efektywne zarządzanie przestrzenią.

W praktyce oznacza to, że bot zapamiętuje jedynie najnowsze interakcje, ignorując te, które są już historycznie mniej istotne. Taki model działania pozwala uniknąć nadmiernego obciążenia pamięci i zapewnia, że decyzje podejmowane przez system są oparte na aktualnym kontekście.

Dzięki zastosowaniu modeli LLM oraz dodatkowych narzędzi, takich jak Mineflayer i mcdata, bot jest w stanie realizować zarówno proste, jak i złożone zadania w grze Minecraft. Połączenie szybkiego modelu LLaMA 3 70B oraz wszechstronnego ChatGPT-4 umożliwia elastyczne i efektywne sterowanie botem w dynamicznym środowisku gry. System pamięci konwersacyjnej dodatkowo wspiera podejmowanie decyzji, zwiększając skuteczność działania bota.

6. Tworzenie i korzystanie z poradników

Bot został zaprojektowany tak, aby mógł korzystać z dynamicznie tworzonych poradników wspierających realizację zadań. Poradniki te są generowane na podstawie zaawansowanych modeli językowych oraz dostępu do internetu. Proces korzystania z nich obejmuje kilka kluczowych etapów, które zapewniają efektywne wykonanie zadania.

6.1. Wykorzystanie istniejących poradników

Kiedy bot otrzymuje nowe zadanie, pierwszy krok polega na sprawdzeniu, czy odpowiedni poradnik znajduje się już w jego bibliotece. Do tego celu wykorzystywany jest lekki model językowy, który analizuje bibliotekę poradników, identyfikując potencjalnie pasujące dokumenty. Jeśli poradnik zostanie odnaleziony, bot realizuje opisane w nim kroki sekwencyjnie, aż do momentu zakończenia zadania.

6.2. Tworzenie nowych poradników

Jeżeli biblioteka nie zawiera poradnika dotyczącego zadanego problemu, bot ocenia, czy konieczne jest stworzenie nowego. Decyzja ta opiera się na analizie złożoności zadania – poradnik jest generowany tylko wtedy, gdy wykonanie wymaga więcej niż jednej akcji.

W procesie tworzenia poradnika bot wywołuje model GPT-4o, który dzięki dostępowi do internetu (za pomocą biblioteki Selenium) oraz własnej wiedzy jest w stanie dokładnie przemyśleć sposób rozwiązania problemu. W trakcie tego procesu model może wejść w tryb "myślenia", aby szczegółowo zaplanować wszystkie kroki potrzebne do realizacji zadania. Gotowy poradnik zostaje zapisany jako plik tekstowy, z odpowiednio dobranym tytułem, który ułatwia jego rozpoznanie w przyszłości.

6.3. Realizacja zadań na podstawie poradnika

Po stworzeniu lub odnalezieniu odpowiedniego poradnika bot przechodzi do jego realizacji. Proces ten obejmuje wykonywanie poszczególnych kroków zgodnie z instrukcją, przy jednoczesnym monitorowaniu rezultatów swoich działań. Bot analizuje, czy poszczególne etapy zostały pomyślnie ukończone, a w przypadku napotkania trudności wprowadza korekty w swoich działaniach.

Gdy bot zauważy, że wszystkie kroki zostały zrealizowane i zadanie jest ukończone, automatycznie przerywa wykonywanie poradnika, wydając odpowiednią komendę końcową.

6.4. Przykład wygenerowanego poradnika

Przykładowy poradnik wygenerowany przez system może wyglądać następująco:

> Aby stworzyć diamentowy kilof w Minecraft, nawet zaczynając od zera, musisz przejść przez kilka etapów:
>
> 1. Zbierz drewno, ścinając drzewa, co dostarczy ci kłody (*log*).
> 2. Przetwórz kłody na deski (*planks*), które posłużą do stworzenia patyków (*sticks*) oraz stołu rzemieślniczego (*crafting table*).
> 3. Użyj stołu rzemieślniczego, aby z desek i patyków wykonać drewniany kilof (*wooden pickaxe*).
> 4. Z drewnianym kilofem wydobądź kamień (*stone*), który po zebraniu stanie się brukiem (*cobblestone*).
> 5. Z bruku i patyków stwórz kamienny kilof (*stone pickaxe*).
> 6. Następnie użyj kamiennego kilofa, aby wydobyć rudę żelaza (*iron ore*).
> 7. Zbuduj piec (*furnace*) z bruku i przetop rudę żelaza na sztabki żelaza (*iron ingots*).
> 8. Wykorzystaj sztabki żelaza i patyki do stworzenia żelaznego kilofa (*iron pickaxe*).
> 9. Z żelaznym kilofem poszukaj rudy diamentu (*diamond ore*), którą można znaleźć na niższych poziomach świata, zwykle między poziomami 5 a 12.
> 10. Wydobądź co najmniej trzy diamenty.
> 11. Na koniec, używając stołu rzemieślniczego, połącz trzy diamenty z dwoma patykami, aby stworzyć diamentowy kilof (*diamond pickaxe*).

Taki poradnik jest szczegółowy i pozwala na stworzenie zaawansowanego narzędzia, zaczynając od zera. Co istotne, bot jest zaprojektowany tak, aby rozpoznawać istniejące zasoby. Na przykład, jeśli gracz posiada już żelazny kilof, bot może pominąć kroki związane z jego tworzeniem i rozpocząć realizację od odpowiedniego etapu.

Dynamiczne tworzenie i wykorzystywanie poradników pozwala botowi na efektywne rozwiązywanie zadań o różnym poziomie złożoności. Dzięki temu system staje się bardziej elastyczny i zdolny do adaptacji w różnych warunkach, zapewniając użytkownikowi wsparcie zarówno w prostych, jak i złożonych wyzwaniach w grze Minecraft.

7. Budowanie przez bota

Budowanie struktur w środowisku gry *Minecraft* stanowi jedno z najbardziej wymagających zadań, przed jakimi staje bot. Proces ten wymaga precyzyjnego zarządzania informacjami oraz umiejętności odwzorowywania skomplikowanych układów bloków w świecie gry. Jednym z kluczowych wyzwań jest brak natywnej zdolności modeli językowych do analizowania danych wizualnych, co dodatkowo komplikuje realizację takich zadań. Złożoność środowiska gry, różnorodność dostępnych materiałów oraz potencjalnie skomplikowane projekty budowlane czynią to zadanie szczególnie trudnym.

Początkowe próby realizacji budowania wykazały, że konieczne jest opracowanie strategii minimalizującej zależność od analizy wizualnej. Z tego względu zaimplementowano podejście polegające na skanowaniu istniejących budynków i tworzeniu ich szczegółowych opisów w formacie JSON. Dzięki temu bot może budować precyzyjnie, wykorzystując dane strukturalne zamiast bezpośrednich obserwacji wizualnych.

7.1. Proces skanowania i odtwarzania budynków

Aby umożliwić botowi efektywne budowanie, opracowano system skanowania budynków znajdujących się w grze. Proces ten polega na analizie konstrukcji w celu zrozumienia jej wymiarów, układu bloków oraz zastosowanych materiałów. Wynikiem skanowania jest plik JSON, który zawiera szczegółowy opis budynku, w tym jego:

- wymiary,
- położenie bloków względem siebie,
- informacje o materiałach użytych do budowy,
- relatywne położenie względem bota, co umożliwia odtworzenie struktury w dowolnym miejscu świata gry.

W procesie skanowania bot korzysta z danych dostarczanych przez bibliotekę *Minecraft Data*, która zawiera szczegółowe informacje o wszystkich dostępnych w grze materiałach.

Dzięki temu możliwe jest precyzyjne przypisanie każdego bloku do odpowiedniej kategorii materiałowej, co z kolei pozwala na wierne odwzorowanie konstrukcji. Przykładowe 2 bloki ze skanu odtworzonego budynku AGH (Grafika 7.1) w Minecrafcie wygląda tak:


```
{
   "relativePosition": {
     "x": -5,
     "y": 0,
     "z": -4
   },
   "id": 563,
   "name": "smooth_stone"
 },
 {
   "relativePosition": {
     "x": -5,
     "y": 1,
     "z": -4
   },
   "id": 7,
   "name": "polished_andesite"
 }
```


Po zapisaniu danych bot może odtworzyć budynek, korzystając wyłącznie z zapisanych informacji. Proces ten obejmuje:

1. Sprawdzenie, czy w aktualnym ekwipunku znajdują się wszystkie wymagane materiały. W przypadku ich braku bot może kontynuować budowanie, pomijając miejsca, dla których nie ma dostępnych zasobów – zależy to od wybranej konfiguracji.

2. Sortowanie bloków według współrzędnej pionowej, co pozwala rozpocząć budowę od najniższych elementów konstrukcji.

3. Tworzenie poszczególnych pięter struktury poprzez ustawianie bloków w odpowiednich miejscach.

4. Usuwanie bloków pomocniczych, które zostały tymczasowo użyte do dotarcia do wyższych partii budynku. Bloki te są łatwe do zidentyfikowania, ponieważ występują w odtworzonej strukturze, ale nie były obecne w oryginalnym projekcie.

Grafika. 7.1. Budynek AGH na naszym prywantym własnym serwerze. zrzut ekranu, źródło własne

Dzięki takiemu podejściu proces budowania staje się bardziej efektywny, a jednocześnie minimalizowane są błędy wynikające z ograniczeń modelu językowego. Taka strategia pozwala na dokładne odwzorowanie nawet bardzo skomplikowanych struktur.

7.2. Znaczenie zastosowanego podejścia

Zastosowane podejście, oparte na skanowaniu i odtwarzaniu budynków, pozwala na skuteczne przezwyciężenie ograniczeń modeli językowych w analizie wizualnej. Pliki JSON generowane podczas skanowania stanowią uniwersalny format, który umożliwia:

– łatwą modyfikację projektów,

– kopiowanie struktur,

– przenoszenie budynków w inne miejsca w świecie gry.

Rozwiązanie to otwiera nowe możliwości w automatyzacji budowania w środowiskach symulacyjnych i może być zaadaptowane do innych gier czy projektów. Dodatkowo, jest ono analogiczne do metod stosowanych w robotyce, gdzie systemy sterujące robotami wykorzystują szczegółowe mapy i modele otoczenia. Dzięki temu projekt ten może przyczynić się do rozwoju technologii integrujących modele językowe z systemami zdolnymi do percepcji i manipulacji zarówno w rzeczywistości wirtualnej, jak i fizycznej.

8. Podstawowe mechaniki sterowania botem

Bot został zaprojektowany w taki sposób, aby obsługiwać podstawowe mechaniki gry *Minecraft*, takie jak walka, spanie, korzystanie z narzędzi, czy wykonywanie prostych czynności w świecie gry. Dzięki zastosowaniu frameworka *Mineflayer* bot może reagować na dynamiczne sytuacje w grze oraz wykonywać złożone sekwencje działań w sposób optymalny i precyzyjny.

8.1. Mechanika walki

Walka jest kluczowym elementem funkcjonowania bota w środowisku gry. Bot został wyposażony w zaawansowane mechanizmy umożliwiające skuteczne eliminowanie przeciwników, zarówno wrogich mobów, jak i innych graczy.

- Bot jest w stanie automatycznie wykryć wrogiego moba w swojej okolicy i podjąć decyzję o ataku.
- Możliwe jest wydanie polecenia, aby bot aktywnie wyszukał konkretnego przeciwnika lub upolował zwierzę danego typu.
- Bot optymalizuje czas ataku, uderzając w idealnych odstępach, aby maksymalizować obrażenia. Uwzględnia zasięg swojej broni i inicjuje atak dokładnie w momencie, gdy cel znajduje się w zasięgu.
- W przypadku posiadania broni i zbroi, bot automatycznie wyposaża się w nie przed rozpoczęciem walki, co zwiększa jego skuteczność i odporność na obrażenia.

Dzięki takim mechanizmom bot może prowadzić walkę w sposób efektywny i zgodny z najlepszymi praktykami w grze, zapewniając graczowi przewagę w starciu z przeciwnikami.

8.2. Korzystanie z narzędzi i obiektów interaktywnych

Bot potrafi w pełni korzystać z interaktywnych obiektów dostępnych w grze, takich jak stół rzemieślniczy czy piec. Wspiera to realizację zadań wymagających wytwarzania przedmiotów lub przetwarzania surowców.

- **Korzystanie ze stołu rzemieślniczego:** Bot jest w stanie wykonywać przepisy rzemieślnicze, używając odpowiednich materiałów i narzędzi. Może tworzyć zarówno proste przedmioty, jak i bardziej zaawansowane konstrukcje wymagające wielu etapów.
- **Korzystanie z pieca:** Bot potrafi używać pieca zarówno do przetapiania rud, jak i przygotowywania żywności. Dzięki temu może przekształcać surowce w bardziej wartościowe przedmioty, takie jak sztabki metali czy pieczone mięso.
- **Ustawianie obiektów:** Bot potrafi postawić takie przedmioty jak piec czy stół rzemieślniczy w odpowiednim miejscu, co pozwala na ich późniejsze użycie.

8.3. Mechanika snu

Bot został zaprogramowany tak, aby mógł korzystać z funkcji snu w grze. Jest w stanie zlokalizować najbliższe łóżko, a następnie z niego skorzystać, co pozwala na pominięcie nocy lub uniknięcie pojawienia się wrogich mobów. Funkcja ta sprawia, że bot jest bardziej autonomiczny i potrafi dostosować swoje działania do zmieniającego się cyklu dnia i nocy w grze.

8.4. Wykonywanie zadań związanych z ruchem i eksploracją

Bot korzysta z zaawansowanych algorytmu DFS (ang. Depth-First Search) zaimplementowanego w *Mineflayer*, aby znaleźć najkrótszą ścieżkę do celu. Jego funkcja umożliwia:

- lokalizowanie najbliższych materiałów, takich jak rudy, drewno czy inne zasoby,
- unikanie przeszkód na drodze do celu,
- szybkie przemieszczanie się w dużych, złożonych środowiskach gry.

Dzięki temu bot może efektywnie eksplorować świat gry i realizować zadania związane z pozyskiwaniem surowców.

8.5. Wyposażanie i wydobywanie

Bot potrafi automatycznie wyposażać narzędzia oraz wydobywać bloki, co umożliwia realizację różnorodnych zadań w grze. W przypadku konieczności zebrania konkretnego materiału bot:

- wybiera odpowiednie narzędzie z ekwipunku, które pozwala na efektywne wydobycie danego surowca,
- przystępuje do kopania bloków, zaczynając od tych, które są najbliżej,
- kontynuuje pracę do momentu zebrania odpowiedniej ilości materiałów.

Dzięki automatycznemu zarządzaniu ekwipunkiem bot zawsze wykorzystuje najbardziej efektywne rozwiązania w kontekście dostępnych narzędzi i zasobów.

Podstawowe mechaniki zaimplementowane w bocie sprawiają, że może on realizować wiele zadań charakterystycznych dla gry *Minecraft*. Mechanizmy te pozwalają na walkę, korzystanie z narzędzi i obiektów interaktywnych, eksplorację oraz wydobywanie zasobów. Dzięki wykorzystaniu frameworka *Mineflayer* bot jest w stanie reagować na dynamiczne zmiany w środowisku gry oraz podejmować decyzje w czasie rzeczywistym, co czyni go wszechstronnym i efektywnym narzędziem w automatyzacji zadań w *Minecraft*.

9. Podsumowanie projektu

W ramach niniejszego projektu zrealizowano założenia dotyczące stworzenia bota zdolnego do autonomicznego wykonywania różnorodnych zadań w środowisku gry *Minecraft*. Efektem pracy jest system, który może korzystać z wielu mechanik gry oraz adaptować swoje działania w dynamicznie zmieniającym się otoczeniu. Bot został zaprogramowany w sposób umożliwiający wykonywanie zarówno prostych czynności, takich jak wydobywanie surowców czy korzystanie z narzędzi, jak i bardziej złożonych zadań, takich jak odtwarzanie budynków, optymalizacja walki czy realizacja instrukcji zawartych w stworzonych przez niego poradnikach.

9.1. Osiągnięte cele

Główne cele projektu zostały zrealizowane, co pozwala stwierdzić, że bot jest w stanie skutecznie:

- wykonywać czynności związane z podstawowymi mechanikami gry, takimi jak kopanie, budowanie, walka czy korzystanie z pieca i stołu rzemieślniczego,
- odtwarzać skomplikowane struktury w oparciu o zapisane wcześniej opisy w formacie JSON, co otwiera nowe możliwości w zakresie automatyzacji budowania,
- optymalnie korzystać z dostępnych zasobów, dostosowując swoje działania do aktualnego stanu ekwipunku oraz celów zadania,
- przemieszczać się oraz znajdować surowce,
- dynamicznie tworzyć i realizować poradniki, umożliwiając efektywne podejście do złożonych zadań wymagających wieloetapowych działań.

Dzięki zastosowaniu modeli językowych bot jest w stanie podejmować decyzje i planować swoje działania w sposób przypominający ludzkie podejście do rozwiązywania problemów. To pozwoliło stworzyć system, który jest nie tylko funkcjonalny, ale również wszechstronny i elastyczny.

9.2. Potencjał rozwoju i przyszłość modeli językowych

Projekt ten ukazuje potencjał, jaki niosą ze sobą zaawansowane modele językowe w kontekście automatyzacji zadań w grach komputerowych. Choć obecne modele, takie jak *GPT*, radzą sobie dobrze z wieloma zadaniami, istnieją jeszcze wyzwania związane z ich ograniczeniami, szczególnie w kontekście analizy wizualnej czy bardzo skomplikowanych operacji wymagających precyzyjnych kalkulacji, czy też wykonywania bardziej skomplikwoanych poleceń.

Wraz z rozwojem technologii modeli językowych możemy jednak oczekiwać, że boty tego typu będą stawały się coraz bardziej zaawansowane. Przyszłe modele prawdopodobnie będą:

– jeszcze lepiej integrować różne typy danych (tekstowe, wizualne, dźwiękowe), co umożliwi pełniejsze rozumienie środowiska gry,

– bardziej efektywnie planować swoje działania w czasie rzeczywistym, minimalizując błędy i poprawiając jakość podejmowanych decyzji,

– zdolne do wykonywania bardziej kreatywnych i złożonych zadań, co pozwoli na ich zastosowanie w szerokim spektrum gier i symulacji.

9.3. Wnioski końcowe

Projekt ten pokazuje, że zaawansowane modele językowe, mimo swoich obecnych ograniczeń, mają ogromny potencjał w automatyzacji procesów i zadań w wirtualnych środowiskach. Stworzony bot jest przykładem zastosowania sztucznej inteligencji do rozwiązywania rzeczywistych problemów w grze, łącząc zdolność do analizy, planowania i działania.

Wraz z dalszym rozwojem modeli językowych oraz integracji z innymi technologiami, możemy spodziewać się kolejnych przełomów w tej dziedzinie, co otwiera perspektywy nie tylko w grach, ale również w szerszych kontekstach, takich jak robotyka, symulacje czy edukacja.

Bibliografia

[1] A Vaswani. „*Attention is all you need*". W: *Advances in Neural Information Processing Systems* (2017).

[2] Guanzhi Wang i in. „*Voyager: An open-ended embodied agent with large language models*". W: *arXiv preprint arXiv:2305.16291* (2023).

[3] Shunyu Liu i in. „*Odyssey: Empowering Minecraft Agents with Open-World Skills*". W: *arXiv preprint arXiv:2407.15325* (2024).

[4] John Merkin. „*Machine Learning in Minecraft: Proof of Concept for Object Detection Oriented Autonomous Bots in Minecraft*". W: (2023).

[5] Koya Kudo Ian Frank. „*An LLM Chatbot in Minecraft with Educational Applications*". W: ().

[6] Ryan Volum i in. „*Craft an iron sword: Dynamically generating interactive game characters by prompting large language models tuned on code*". W: *Proceedings of the 3rd Wordplay: When Language Meets Games Workshop (Wordplay 2022)*. 2022, s. 25–43.

[7] Gurjeet Singh. „*Leveraging chatgpt for real-time decision-making in autonomous systems*". W: *Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal* 12.2 (2023), s. 101–106.

[8] Miguel Á González-Santamarta i in. „*Using large language models for interpreting autonomous robots behaviors*". W: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2023, s. 533–544.

[9] Ferran Sanchez Llado. „*Controlling Agents Behaviours through LLMs*". 2024.

[10] Sihao Hu i in. „*A survey on large language model-based game agents*". W: *arXiv preprint arXiv:2404.02039* (2024).