Kacper Płachetka

Polsko-Japońska Akademia Technik Komputerowych

Projekt 1 – s30734

Spis treści

Interfejs Stackable

Interfejs **Stackable** definiuje metody do obsługi blocków i ich ilości.

Stałe

- **int minQuantity = 0** - Minimalna ilość przedmiotów w stacku.
- **int maxQuantity = 64** - Maksymalna ilość przedmiotów na stacku.

Metody

int count()

Zwraca wielkość stacku.

Zwraca:

- Liczba przedmiotów w stacku.

int addToStack(int quantity)

Dodaje określoną liczbę przedmiotów do stosu.

Parametry:

- **quantity** - liczba przedmiotów do dodania do stosu.

Zwraca:

- Zwraca liczbę niedodanych elementów do stacku.

void removeFromStack(int quantity, Inventory inventory)

Usuwa określoną liczbę przedmiotów ze stacku i umieszcza je w podanym ekwipunku.

Parametry:

- **quantity** - liczba przedmiotów do usunięcia ze stacku.
- **inventory** - ekwipunek, do którego zostaną przeniesione usunięte przedmioty.

Interfejs Repairable

Interfejs **Repairable** definiuje metodę do naprawy obiektów, które mogą ulec uszkodzeniu lub zużyciu. Interfejs ten może być implementowany przez klasy, które reprezentują naprawialne przedmioty, np. narzędzia, broń, zbroje.

Metody

void repair()

Naprawia obiekt implementujący tę metodę. Implementacja tej metody powinna określać, warunki naprawy obiektu i jakie efekty ona powoduje.

Interfejs Movable

Interfejs **Movable** definiuje metodę, która powinna być zaimplementowana przez klasy reprezentujące obiekty mogące się poruszać. Interfejs ten może być używany w klasach implementujących ruch gracza, zwierząt, ludzi.

Metody

void move()

Porusza obiekt implementujący tę metodę. Konkretna implementacja tej metody powinna określać zmianę koordynatów danego obiektu.

Klasa Block

Klasa **Block** jest abstrakcyjną klasą stanowiącą podstawę dla innych klas bloków w grze. Implementuje interfejs **Stackable**, co pozwala na zarządzanie ilością bloków w stacku. Bloki mogą być umieszczane w świecie gry, zbierane do inwentarza i usuwane z niego.

Pola

- **public String name** - Nazwa bloku.
- **public int blockQuantity** - Ilość bloków w stacku.
- **public int timeToDestroy** - Czas potrzebny na zniszczenie bloku w milisekundach (domyślna wartość to 5000).
- **public ToolsNames bestToDestroy** - Najlepsze narzędzie do niszczenia tego bloku.

Konstruktor

- **public Block()** - Domyślny konstruktor.
- **public Block(String name)** - Konstruktor z nazwą bloku.

Metody

void place()

Umieszcza blok w świecie gry na pozycji gracza, jeśli miejsce jest puste.

void collect(Inventory inventory)

Zbiera blok do ekwipunku. Jeśli w ekwipunku znajduje się już stack tego samego typu bloku, zwiększa jego ilość, w przeciwnym razie dodaje nowy block do inwentarza.

public int count()

Zwraca ilość bloków w stacku.

Zwraca:

- Ilość bloków w stacku.

public int addToStack(int quantity)

Dodaje określoną ilość bloków do stacku.

Parametry:

- **quantity** - ilość bloków do dodania.

Zwraca:

- Ilość bloków, które nie zmieściły się w stacku (jeśli **quantity + blockQuantity** przekracza **Stackable.maxQuantity**).

public void removeFromStack(int quantity, Inventory inventory)

Usuwa określoną ilość bloków ze stacku i aktualizuje ekwipunek.

Parametry:

- **quantity** - ilość bloków do usunięcia.
- **inventory** - inwentarz gracza.

public void removeBlockFromInventory(Inventory inventory)

Usuwa blok z ekwipunku, jeśli jego ilość w stacku osiągnie zero.

Parametry:

- **inventory** - ekwipunek gracza.

public abstract String getColor()

Abstrakcyjna metoda zwracająca kolor bloku. Implementacje konkretnych typów bloków powinny zdefiniować tę metodę.

Zwraca:

- Kolor bloku w postaci **String**.

@Override public String toString()

Zwraca reprezentację bloku jako **String**, zawierającą jego nazwę i ilość.

Zwraca:

- Reprezentację bloku jako **String**.

Klasa None

Klasa **None** rozszerza abstrakcyjną klasę **Block** i reprezentuje puste miejsce w świecie gry. Blok **None** nie ma koloru i jest używany do oznaczania pustych pozycji, które mogą być zajęte przez inne bloki.

Konstruktor

- **public None()** - Domyślny konstruktor klasy **None**.

Metody

@Override public String getColor()

Zwraca pusty ciąg znaków, co oznacza, że blok **None** nie ma przypisanego koloru.

Zwraca:

Pusty ciąg znaków (**""**).

Klasa Dirt

Klasa **Dirt** rozszerza abstrakcyjną klasę **Block** i reprezentuje blok ziemi w grze. Blok ziemi ma określone właściwości, takie jak nazwa, czas niszczenia, najlepsze narzędzie do niszczenia oraz kolor.

Konstruktor

- **public Dirt()**

Inicjalizuje nową instancję bloku ziemi z domyślnymi wartościami:

- **name** - "Dirt"
- **timeToDestroy** - 5000 milisekund
- **bestToDestroy** - **ToolsNames.SHOVEL**
- **blockQuantity** - 1

Metody

@Override public String getColor()

Zwraca kolor bloku ziemi.

Zwraca:

- "green" - Kolor bloku ziemi.

Klasa Stone

Klasa **Stone** rozszerza abstrakcyjną klasę **Block** i reprezentuje blok kamienia w grze. Blok kamienia ma określone właściwości, takie jak nazwa, czas niszczenia, najlepsze narzędzie do niszczenia oraz kolor.

Konstruktor

- **public Stone()**

Inicjalizuje nową instancję bloku kamienia z domyślnymi wartościami:

- **name** - "Stone"
- **timeToDestroy** - 5500 milisekund
- **bestToDestroy** - **ToolsNames.PICKAXE**
- **blockQuantity** - 1

Metody

@Override public String getColor()

Zwraca kolor bloku kamienia.

Zwraca:

"gray" - Kolor bloku kamienia.

Klasa Entity

Klasa **Entity** jest abstrakcyjną klasą bazową dla wszystkich istot w grze. Reprezentuje podstawowe cechy i zachowania, takie jak zdrowie, pozycja, stan (np. bycie podpalonym), a także operacje związane z ruchem i interakcją między jednostkami.

Pola

- **public String name** - Nazwa jednostki.
- **public int health** - Zdrowie jednostki (domyślnie 10).
- **boolean onFire** - Flaga określająca, czy jednostka jest podpalona (domyślnie **false**).
- **public int positionX** - Pozycja X jednostki.
- **public int positionY** - Pozycja Y jednostki.
- **final int offset** - Offset dla pozycji, równy połowie długości świata (**World.world.length / 2**).
- **public static List<Entity> entities** - Lista wszystkich jednostek w grze.

Konstruktor

- **public Entity(String name)**

Parametry:

- **name** - Nazwa jednostki.

Metody

public abstract void hit()

Metoda abstrakcyjna, która musi być zaimplementowana przez klasy dziedziczące, reprezentuje uderzenie w jednostkę.

public void kill(Entity entity)

Zabija jednostkę, ustawiając jej zdrowie na 0 i usuwając ją z listy jednostek.

Parametry:

- **entity** - Jednostka do zabicia.

public boolean isOnFire()

Zwraca, czy jednostka jest podpalona.

Zwraca:

- **true** jeśli jednostka jest podpalona, **false** w przeciwnym razie.

public void setOnFire(Entity entity)

Podpala jednostkę, zmniejszając jej zdrowie o 1 co sekundę przez 3 sekundy.

Parametry:

- **entity** - Jednostka do podpalenia.

public void getStatus()

Wypisuje status jednostki, w tym zdrowie, stan podpalenia i aktualną pozycję.

public void checkHealth()

Sprawdza zdrowie wszystkich jednostek i usuwa martwe jednostki z listy, z wyjątkiem gracza. Jeśli zdrowie gracza wynosi 0, kończy grę.

public void moveUp()

Przesuwa jednostkę w górę, jeśli to możliwe.

public void moveDown()

Przesuwa jednostkę w dół, jeśli to możliwe.

public void moveLeft()

Przesuwa jednostkę w lewo, jeśli to możliwe.

public void moveRight()

Przesuwa jednostkę w prawo, jeśli to możliwe.

public Double distance(Entity e1, Entity e2)

Oblicza dystans pomiędzy dwiema jednostkami.

Parametry:

- **e1** - Pierwsza jednostka.
- **e2** - Druga jednostka.

Zwraca:

- Dystans pomiędzy jednostkami.

public Entity findClosestEntity()

Znajduje najbliższą jednostkę w stosunku do bieżącej jednostki.

Zwraca:

- Najbliższą jednostkę.

public int getPositionX()

Zwraca pozycję X jednostki.

Zwraca:

- Pozycję X jednostki.

public int getPositionY()

Zwraca pozycję Y jednostki.

Zwraca:

- Pozycję Y jednostki.

@Override public String toString()

Zwraca nazwę jednostki jako reprezentację tekstową.

Zwraca:

- Nazwę jednostki.

Klasa Player

Klasa **Player** rozszerza klasę **Entity** i implementuje interfejs **Movable**. Reprezentuje gracza w grze, zarządza jego ruchem, ekwipunkiem oraz interakcją z otoczeniem.

Pola

- **private Object currentInHand** - Obiekt aktualnie trzymany w ręce przez gracza.
- **private int currentIndex** - Indeks aktualnie trzymanego przedmiotu w inwentarzu.
- **private World world** - Odniesienie do obiektu **World**, w którym znajduje się gracz.
- **private JTextArea textArea** - Obszar tekstowy do wyświetlania wiadomości w interfejsie użytkownika.
- **private Inventory inventory** - Ekwipunek gracza.
- **public static int positionX** - Pozycja X gracza.
- **public static int positionY** - Pozycja Y gracza.
- **private final Set<Integer> pressedKeys** - Zestaw klawiszy aktualnie wciśniętych przez gracza.

Konstruktor

- **public Player(String name)**

  Parametry:

  name - Nazwa gracza.

  Ustawia pozycję gracza na środek świata (**World.world.length / 2**) i ustawia domyślnie pierwszy przedmiot w ręce.

- **public Player(String name, World world, Inventory inventory, JTextArea textArea)**

  Inicjalizuje nową instancję gracza z określoną nazwą, światem, inwentarzem i obszarem tekstowym.

  Parametry:

  name - Nazwa gracza.

  world - Świat, w którym znajduje się gracz.

  inventory - Ekwipunek gracza.

  textArea - Obszar tekstowy do wyświetlania wiadomości.

Metody

public void setCurrentInHand(int n)

Ustawia przedmiot trzymany w ręce na podstawie indeksu w ekwipunku.

Parametry:

- **n** - Indeks przedmiotu w ekwipunku.

public void placeBlock()

Umieszcza blok w świecie gry w aktualnej pozycji gracza.

@Override public void hit()

Uderza najbliższą jednostkę w zasięgu, jeśli istnieje.

public void moveUp()

Przesuwa gracza w górę, jeśli to możliwe.

public void moveDown()

Przesuwa gracza w dół, jeśli to możliwe.

public void moveLeft()

Przesuwa gracza w lewo, jeśli to możliwe.

public void moveRight()

Przesuwa gracza w prawo, jeśli to możliwe.

public void keyPressed(KeyEvent e)

Dodaje kod wciśniętego klawisza do zestawu wciśniętych klawiszy.

Parametry:

- **e** - Zdarzenie klawisza.

public void keyReleased(KeyEvent e)

Usuwa kod zwolnionego klawisza z zestawu wciśniętych klawiszy.

Parametry:

- **e** - Zdarzenie klawisza.

@Override public void move()

Przesuwa gracza na podstawie aktualnie wciśniętych klawiszy (W, A, S, D).

public void mine()

Rozpoczyna proces wydobycia bloku w aktualnej pozycji gracza.

public void inspectBlock()

Wypisuje informacje o bloku, na którym stoi gracz.

private void appendToConsole(String message)

Dodaje wiadomość do obszaru tekstowego i przewija do końca.

Parametry:

- **message** - Wiadomość do wyświetlenia.

@Override public int getPositionX()

Zwraca pozycję X gracza.

Zwraca:

- Pozycję X gracza.

@Override public int getPositionY()

Zwraca pozycję Y gracza.

Zwraca:

- Pozycję Y gracza.

public int getCurrentIndex()

Zwraca indeks aktualnie trzymanego przedmiotu w ekwipunku.

Zwraca:

- Indeks aktualnie trzymanego przedmiotu.

Klasa Creeper

Klasa **Creeper** rozszerza klasę **Entity** i implementuje interfejs **Movable**. Reprezentuje jednostkę typu Creeper, która losowo porusza się po świecie i atakuje gracza, gdy jest wystarczająco blisko.

Pola

- **Random random** - Generator liczb losowych używany do losowego poruszania się.
- **private Timer timer** - Timer odpowiedzialny za okresowe poruszanie się Creepera.

Konstruktor

- **public Creeper(String name)**

  Inicjalizuje nową instancję Creepera z określoną nazwą.

  Dodaje Creepera do listy jednostek **Entity.entities**.

  Ustawia początkową pozycję Creepera na **(100, 102)**.

  Uruchamia automatyczne poruszanie się Creepera.

Parametry:

- **name** - Nazwa Creepera.

Metody

@Override public void hit()

Sprawdza, czy gracz znajduje się w zasięgu uderzenia (dystans <= 2). Jeśli tak, Creeper przestaje się poruszać i po krótkiej przerwie eksploduje, zadając obrażenia graczowi (6 punktów zdrowia). Jeśli gracz oddali się na większą odległość, Creeper ponownie zaczyna się poruszać.

@Override public void move()

Losowo porusza Creepera w jednym z czterech kierunków: w górę, w dół, w lewo lub w prawo.

private void startMoving(Boolean isWalking)

Rozpoczyna lub zatrzymuje automatyczne poruszanie się Creepera. Timer wywołuje metodę **move()** co sekundę.

Parametry:

- **isWalking** - Flaga określająca, czy Creeper ma się poruszać (**true**) lub zatrzymać (**false**).

@Override public void checkHealth()

Sprawdza stan zdrowia Creepera. Jeśli zdrowie jest równe lub mniejsze od 0, zatrzymuje timer i usuwa Creepera z listy jednostek. W przeciwnym razie wywołuje metodę **checkHealth()** z klasy nadrzędnej.

Klasa Pig

Klasa **Pig** rozszerza klasę **Entity** i implementuje interfejs **Movable**. Reprezentuje jednostkę typu Pig, która losowo porusza się po świecie i atakuje Creepery, gdy znajdą się wystarczająco blisko.

Pola

- **Random random** - Generator liczb losowych używany do losowego poruszania się.
- **private Timer timer** - Timer odpowiedzialny za okresowe poruszanie się świni.

Konstruktor

- **public Pig(String name)**

  Inicjalizuje nową instancję Pig'a z określoną nazwą.

  Dodaje Pig do listy jednostek **Entity.entities**.

  Ustawia początkową pozycję Pig na **(107, 100)**.

  Uruchamia automatyczne poruszanie się Pig.

Parametry:

- **name** - Nazwa świnki.

Metody

@Override public void hit()

Sprawdza, czy Creeper znajduje się w zasięgu uderzenia (dystans <= 1). Jeśli tak, Pig zadaje obrażenia Creeperowi (1 punkt zdrowia) i czeka 2 sekundy przed kolejnym atakiem.

@Override public void move()

Losowo porusza Pig'a w jednym z czterech kierunków: w górę, w dół, w lewo lub w prawo.

private void startMoving(Boolean isWalking)

Rozpoczyna lub zatrzymuje automatyczne poruszanie się Pig'a. Timer wywołuje metodę **move()** co sekundę.

Parametry:

- **isWalking** - Flaga określająca, czy Pig ma się poruszać (**true**) lub zatrzymać (**false**).

@Override public void checkHealth()

Sprawdza stan zdrowia Pig. Jeśli zdrowie jest równe lub mniejsze od 0, zatrzymuje timer i usuwa Pig z listy jednostek. W przeciwnym razie wywołuje metodę **checkHealth()** z klasy nadrzędnej.

Klasa Tool

Klasa **Tool** jest abstrakcyjną klasą bazową dla wszystkich narzędzi w grze. Reprezentuje narzędzia używane przez graczy do interakcji z blokami i jednostkami.

Pola

- **int damage** - Łączne obrażenia, jakie narzędzie może zadać przed zniszczeniem.
- **public int damageToEntity** - Obrażenia zadawane jednostkom przy użyciu tego narzędzia.
- **public ToolsNames toolsNames** - Nazwa narzędzia.
- **ToolMaterials toolMaterials** - Materiał, z którego wykonane jest narzędzie.

Konstruktory

- **public Tool()**

  Konstruktor domyślny, nie wykonuje żadnych operacji.

- **public Tool(ToolMaterials toolMaterials)**

  Inicjalizuje nową instancję narzędzia z określonym materiałem.

  Ustawia właściwości narzędzia na podstawie wybranego materiału.

Parametry:

- **toolMaterials** - Materiał narzędzia.

Metody

@Override public abstract void repair()

Metoda abstrakcyjna do naprawy narzędzia. Musi być zaimplementowana przez klasy dziedziczące.

@Override public String toString()

Zwraca reprezentację tekstową narzędzia, zawierającą materiał i nazwę narzędzia.

Zwraca:

- Tekstowa reprezentacja narzędzia.

Enum ToolsNames

ToolsNames to typ wyliczeniowy reprezentujący nazwy różnych rodzajów narzędzi w grze.

Wartości

- **AXE** - Topór.
- **HOE** - Motyka.
- **PICKAXE** - Kilof.
- **SHOVEL** - Łopata.
- **SWORD** - Miecz.

Enum ToolMaterials

ToolMaterials to typ wyliczeniowy reprezentujący różne materiały, z których wykonane są narzędzia w grze.

Wartości

- **WOOD** - Drewno.
- **STONE** - Kamień.
- **GOLD** - Złoto.
- **IRON** - Żelazo.
- **DIAMOND** - Diament.

Klasa Pickaxe

Pickaxe jest klasą reprezentującą narzędzie typu kilof w grze. Dziedziczy po klasie **Tool** i implementuje metodę **repair()**.

Konstruktor

- **public Pickaxe(ToolMaterials toolMaterials)**

  Inicjalizuje nową instancję kilofa z określonym materiałem.

  Ustawia nazwę narzędzia na **PICKAXE**.

Parametry:

- **toolMaterials** - Materiał, z którego wykonany jest kilof.

Metody

@Override public void repair()

Metoda **repair()** służy do naprawy narzędzia. Użytkownik musi podać indeks innego przedmiotu z ekwipunku, który zostanie użyty do naprawy. Jeśli podany indeks odpowiada przedmiotowi typu **Pickaxe**, to ten przedmiot jest usuwany z ekwipunku, a aktualny kilof jest naprawiany.

Klasa Shovel

Shovel jest klasą reprezentującą narzędzie typu łopata w grze. Dziedziczy po klasie **Tool** i implementuje metodę **repair()**.

Konstruktor

- **public Shovel(ToolMaterials toolMaterials)**

  Inicjalizuje nową instancję łopaty z określonym materiałem.

  Ustawia nazwę narzędzia na **SHOVEL**.

Parametry:

- **toolMaterials** - Materiał, z którego wykonana jest łopata.

Metody

@Override public void repair()

Metoda **repair()** służy do naprawy narzędzia. Użytkownik musi podać indeks innego przedmiotu z ekwipunku, który zostanie użyty do naprawy. Jeśli podany indeks odpowiada przedmiotowi typu **Shovel**, to ten przedmiot jest usuwany z ekwipunku, a aktualna łopata jest naprawiana.

Klasa Sword

Sword jest klasą reprezentującą narzędzie typu miecz w grze. Dziedziczy po klasie **Tool** i implementuje metodę **repair()**.

Konstruktor

- **public Sword(ToolMaterials toolMaterials)**

  Inicjalizuje nową instancję miecza z określonym materiałem.

  Ustawia nazwę narzędzia na **SWORD**.

  Zwiększa obrażenia zadawane jednostkom przez miecz.

Parametry:

- **toolMaterials** - Materiał, z którego wykonany jest miecz.

Metody

@Override public void repair()

Metoda **repair()** służy do naprawy narzędzia. Użytkownik musi podać indeks innego przedmiotu z ekwipunku, który zostanie użyty do naprawy. Jeśli podany indeks odpowiada przedmiotowi typu **Sword**, to ten przedmiot jest usuwany z ekwipunku, a aktualny miecz jest naprawiany.

Klasa World

World reprezentuje świat gry, który składa się z bloków o różnych typach.

Pola

- **public static final Block[][] world**: Dwuwymiarowa tablica bloków, która reprezentuje świat gry.
- **final int offset**: Przesunięcie świata, aby możliwe było użycie ujemnych współrzędnych.

Konstruktor

- **public World()**: Tworzy nowy obiekt klasy **World**. Wypełnia świat blokami typu **Dirt**, a następnie losowo rozmieszcza bloki typu **Stone**.

Klasa Inventory

Inventory reprezentuje ekwipunek gracza w grze.

Pola

- **public static Object[] inventory**: Tablica przedmiotów w ekwipunku.
- **private JTextArea console**: Obszar tekstowy służący do wyświetlania komunikatów o akcjach w ekwipunku.

Konstruktor

- **public Inventory(JTextArea console)**: Tworzy nowy ekwipunek z podanym obszarem tekstowym do wyświetlania komunikatów.

Metody

public void showInventoryOptions()

Metoda **showInventoryOptions()** wyświetla menu opcji dla ekwipunku, w którym gracz może wybrać akcje takie jak usuwanie przedmiotów z ekwipunku, losowe usunięcie przedmiotu lub zamianę miejscami dwóch przedmiotów.

public void showInventory()

Metoda **showInventory()** wyświetla aktualny stan ekwipunku w formie listy.

public boolean isInInventory(Object object)

Metoda **isInInventory(Object object)** sprawdza, czy dany przedmiot znajduje się w ekwipunku.

public boolean isInventoryFull()

Metoda **isInventoryFull()** sprawdza, czy ekwipunek jest pełny.

public void placeInInventory(Object object)

Metoda **placeInInventory(Object object)** dodaje przedmiot do ekwipunku.

public void removeFromInventory(int location)

Metoda **removeFromInventory(int location)** usuwa przedmiot z ekwipunku na podanej pozycji.

public void removeFromInventory()

Metoda **removeFromInventory()** usuwa losowy przedmiot z ekwipunku.

public void replaceInInventory(int a, int b)

Metoda **replaceInInventory(int a, int b)** zamienia miejscami dwa przedmioty w ekwipunku.

Klasa Game

Klasa **Game** reprezentuje główny panel gry, gdzie zaimplementowana jest logika gry, interfejs użytkownika oraz renderowanie grafiki. Wykorzystuje komponenty Swing do graficznego interfejsu użytkownika i renderowania świata gry.

Pola:

- **TILE_SIZE**: Stała całkowita reprezentująca rozmiar każdego bloku w pikselach.
- **VIEW_SIZE**: Stała całkowita reprezentująca liczbę widocznych bloków na ekranie w obu kierunkach.
- **world**: Instancja klasy **World** reprezentująca świat gry.
- **player**: Instancja klasy **Player** reprezentująca postać gracza.
- **inventory**: Instancja klasy **Inventory** reprezentująca ekwipunek gracza.
- **playerImage**: Obiekt typu **BufferedImage** reprezentujący obraz postaci gracza.
- **creeperImage**: Obiekt typu **BufferedImage** reprezentujący obraz stwora creeper.
- **pigImage**: Obiekt typu **BufferedImage** reprezentujący obraz stwora świńskiego.
- **console**: Obiekt typu **JTextArea** używany do wyświetlania komunikatów i interakcji z graczem.

Konstruktor:

- **Game()**: Inicjuje grę poprzez utworzenie świata, konfigurację interfejsu użytkownika oraz ładowanie obrazów dla gracza i stworzeń. Ustawia także preferowany rozmiar panelu gry i wywołuje metodę **setupUI()**.

Metody:

- **setupUI()**: Konfiguruje interfejs użytkownika poprzez tworzenie komponentów Swing, takich jak przyciski dla różnych akcji oraz konsolę do wyświetlania komunikatów. Dodaje także nasłuchiwacze zdarzeń do obsługi interakcji użytkownika.

- **drawWorld(Graphics g)**: Rysuje świat gry poprzez iterację po widocznej części świata i renderowanie każdego bloku oraz stworzenia zgodnie z tym.
- **paintComponent(Graphics g)**: Przesłania metodę **paintComponent** klasy **JPanel** w celu dostosowania renderowania panelu gry.
- **main(String[] args)**: Punkt wejścia do gry, tworzący instancję klasy **Gra** w celu rozpoczęcia gry.

Instrukcja uruchomienia gry i jej obsługi

Grę należy odpalić w klasie **Game()** i przejść do okienka które się otworzy

Jak grać:

Poruszanie odbywa się za pomocą klawiszy:
W – poruszanie się do góry.
S – poruszanie się do dołu.
A – poruszanie się w lewo.
D – poruszanie się w prawo.

Poniżej mapy gry możemy znaleźć wiele przycisków do wykorzystania w grze:
Inventory Options – Wyświetli się okienko z opcjami dotyczącego twojego ekwipunku
Show Inventory – Wyświetli się okienko z listą rzeczy w twoim ekwipunku
Hit – Uderz najbliższą postać
Mine – Wykop blok na którym stoisz
Inspect – Zobacz na jakim bloku obecnie stoisz
Place Block – Połóż blok który obecnie trzymasz w ręce
Change Item – Wymień przedmiot który trzymasz w ręku
Remove from stack – Wyrzuć z obecnie trzymanego bloku ilość bloków których chcesz się pozbyć

Zasady testowania

- Jeśli kliknąłeś przycisk „Mine” nie poruszaj się
- Poczekaj aż operacja spod przycisku „Mine” się zakończy zanim znowu jej użyjesz
- Na początku kod można dodać różne dostępne przedmioty i postacie
- Upewnij się, że wszystkie ikony postaci są w odpowiednim folderze by gra zachowała funkcjonalność

Problemy podczas pisania gry

Implementacja ilości bloków, gdzie przechowywać – Uznałem, że najlogiczniejszym miejscem będzie klasa Block

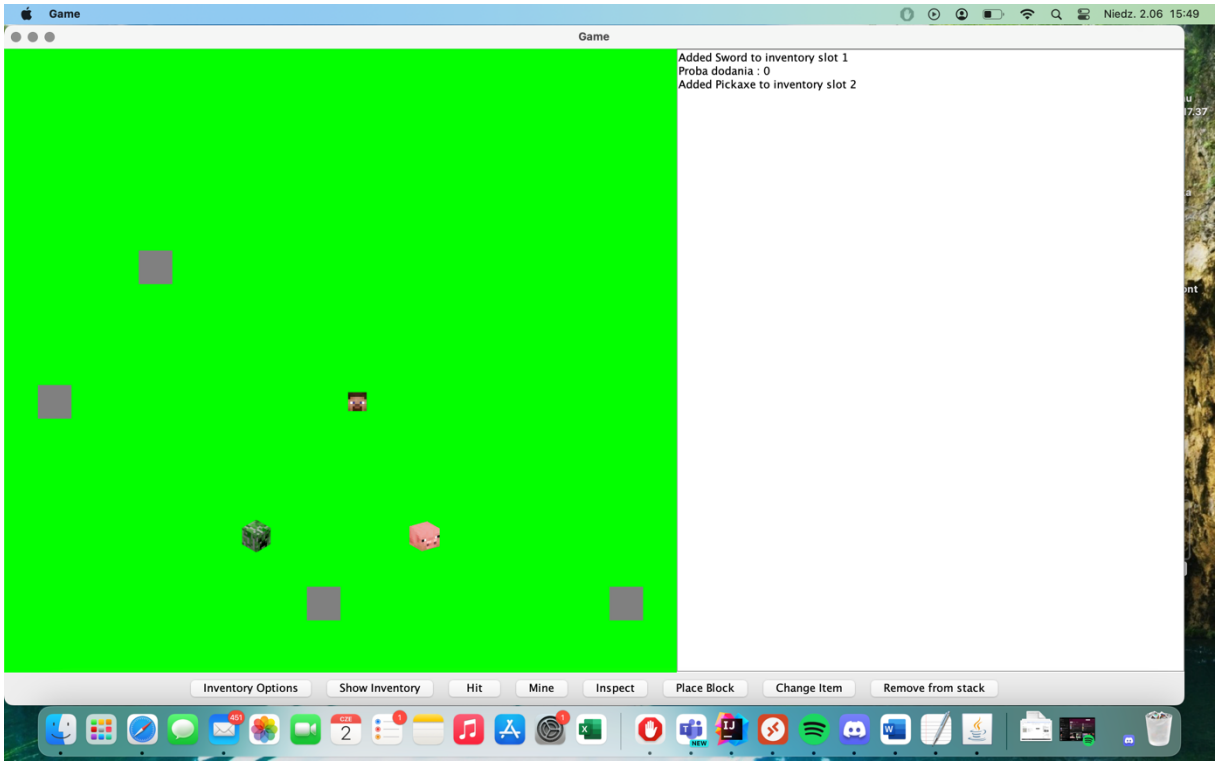
Zapisywanie położenia gracza – statyczne pola w klasie Player, daje to łatwiejsze działanie na nich

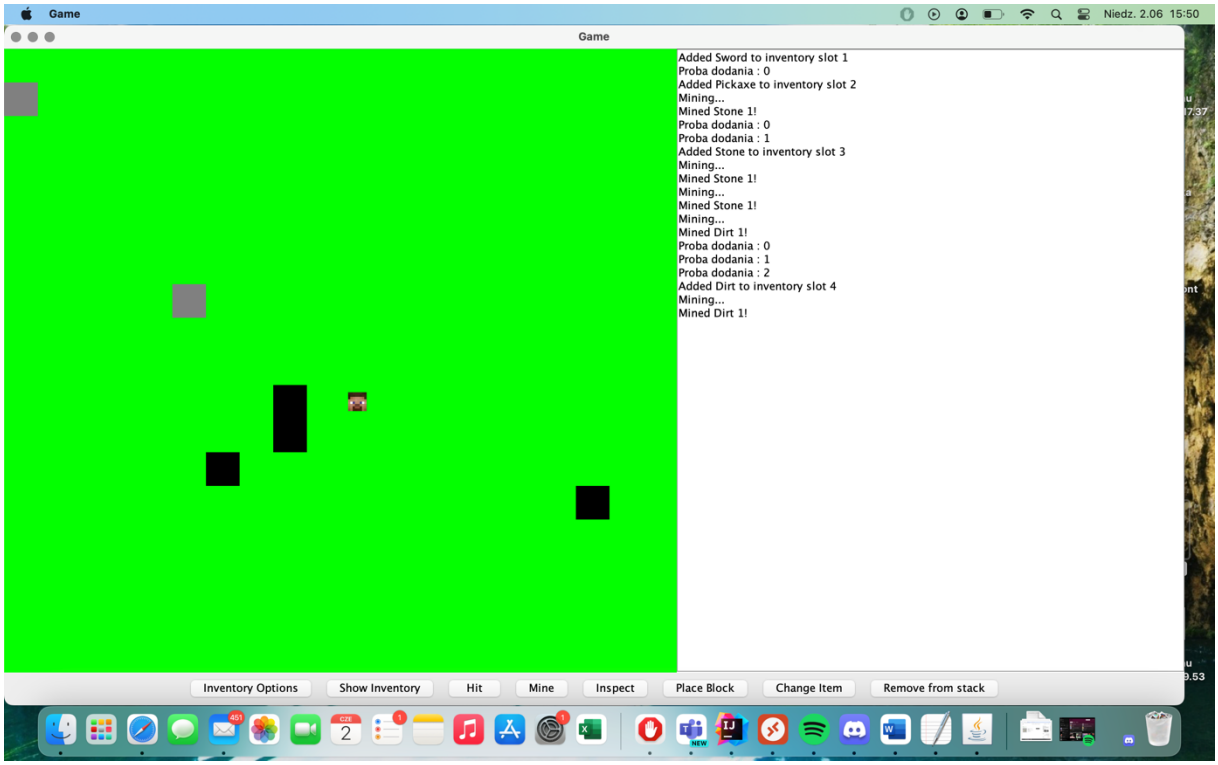
Zadawanie obrażeń – if sprawdzający klasę obiektu używający instaceof

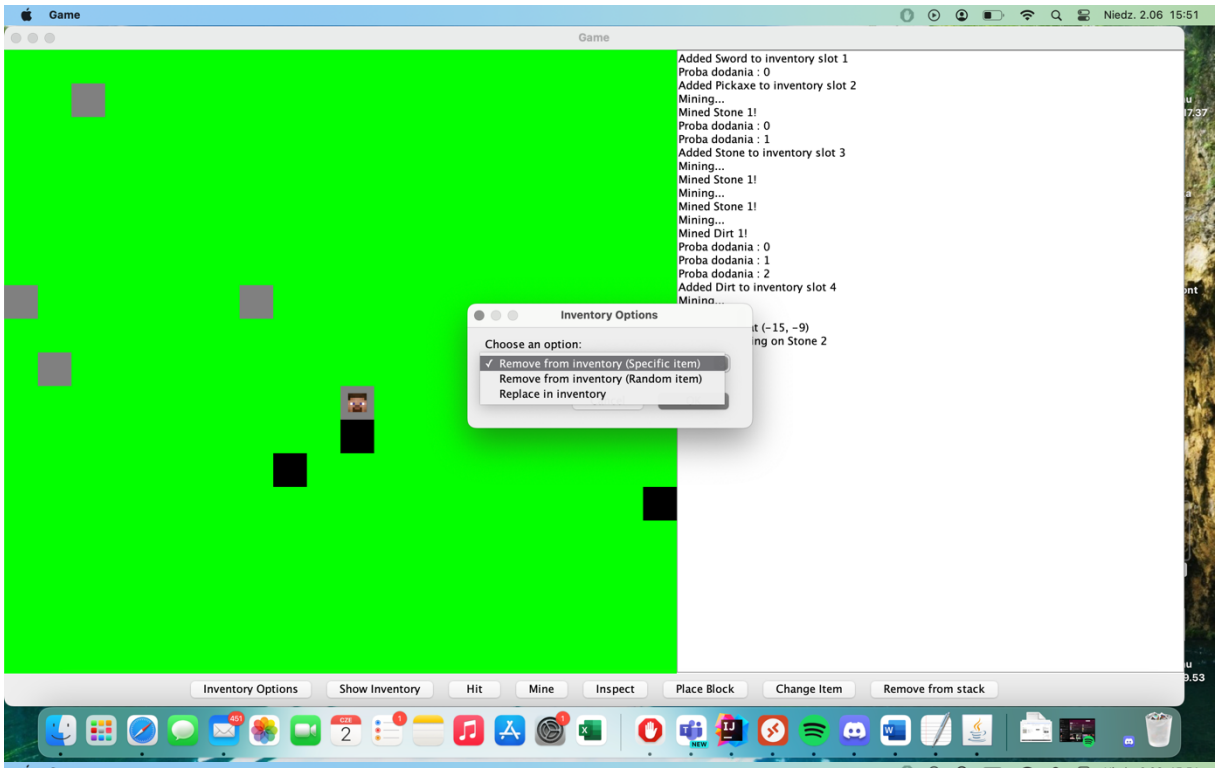
Po usunięciu wszystkich blokow jak zrobić by usunęły się tez z ekwipunku – dodanie metody przy kładzeniu blokow na mapie

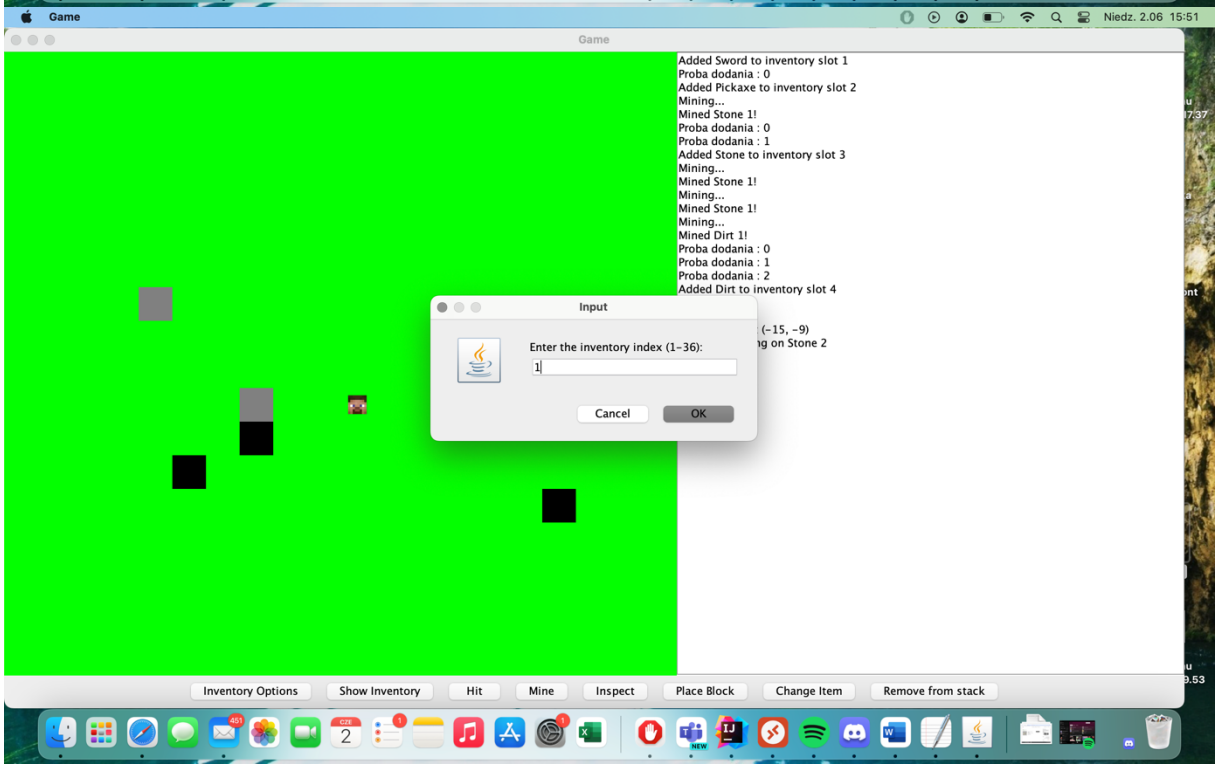
Po użyciu metody kładącej bloki na mapie wartość ilości bloków zmieniała się na -1 – Podczas kolejnego szukania błędu i pisania na nowo metody okazało się, że w klasach dziedziczących po klasie Block metoda place() była nadpisywana i błędem było działanie które, zamiast „-=” miało „=-”

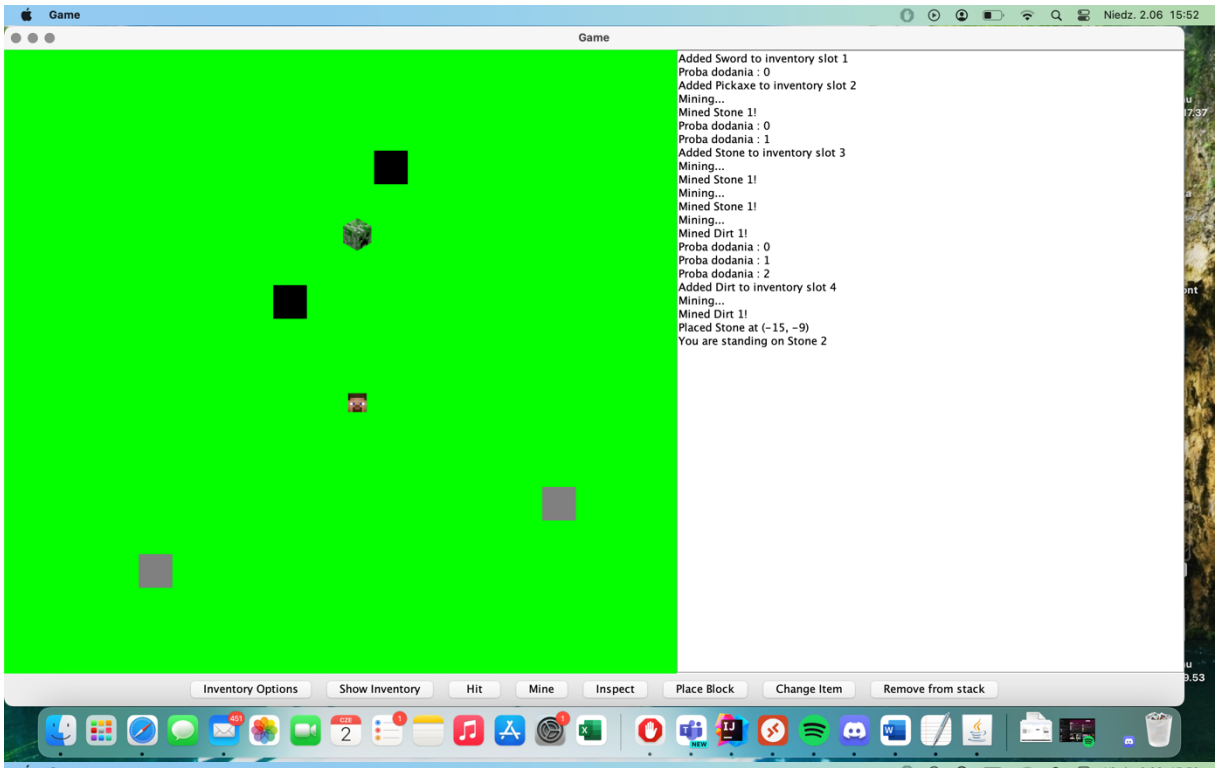
Zrzuty ekranu

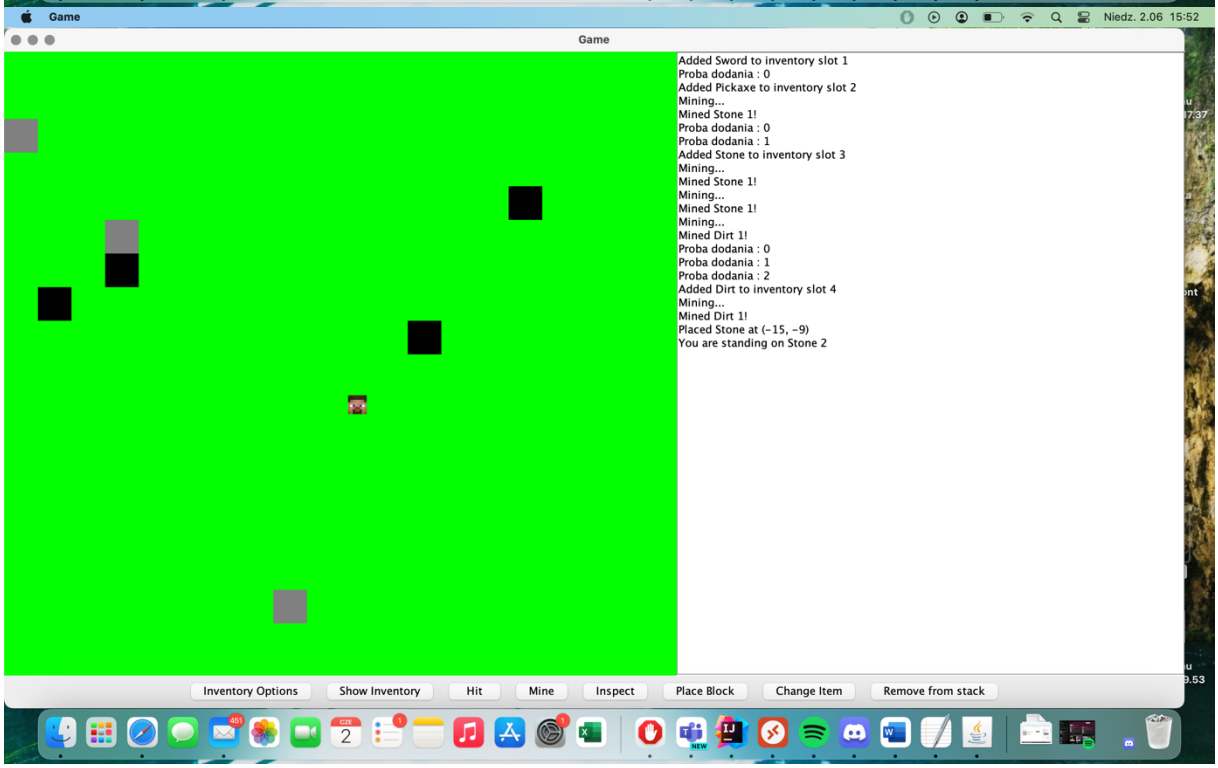


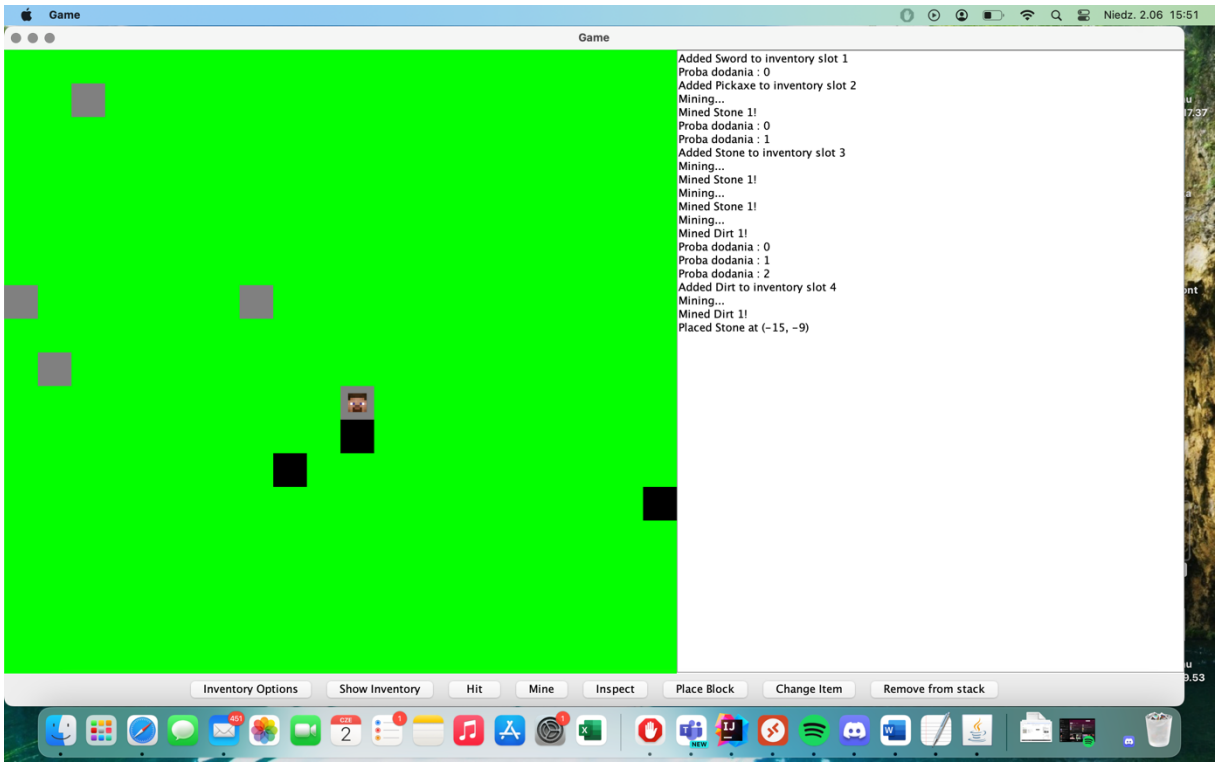



Game
Niedz. 2.06 15:51
Game
Added Sword to inventory slot 1
Proba dodania : 0
Added Pickaxe to inventory slot 2
Mining...
Mined Stone 1!
Proba dodania : 0
Proba dodania : 1
Added Stone to inventory slot 3
Mining...
Mined Stone 1!
Mining...
Mined Stone 1!
Mining...
Mined Dirt 1!
Proba dodania : 0
Proba dodania : 1
Proba dodania : 2
Added Dirt to inventory slot 4
Inventory Options
Choose an option:
Remove from inventory (Specific item)
Remove from inventory (Random item)
Replace in inventory
Inventory Options
Show Inventory
Hit
Mine
Inspect
Place Block
Change Item
Remove from stack


Game
Niedz. 2.06 15:51
Game
Added Sword to inventory slot 1
Proba dodania : 0
Added Pickaxe to inventory slot 2
Mining...
Mined Stone 1!
Proba dodania : 0
Proba dodania : 1
Added Stone to inventory slot 3
Mining...
Mined Stone 1!
Mining...
Mined Stone 1!
Mining...
Mined Dirt 1!
Proba dodania : 0
Proba dodania : 1
Proba dodania : 2
Added Dirt to inventory slot 4
Input
Enter the inventory index (1-36):
1
Cancel
OK
Inventory Options
Show Inventory
Hit
Mine
Inspect
Place Block
Change Item
Remove from stack


Game
Niedz. 2.06 15:52
Game
Added Sword to inventory slot 1
Proba dodania : 0
Added Pickaxe to inventory slot 2
Mining...
Mined Stone 1!
Proba dodania : 0
Proba dodania : 1
Added Stone to inventory slot 3
Mining...
Mined Stone 1!
Mining...
Mined Stone 1!
Mining...
Mined Dirt 1!
Proba dodania : 0
Proba dodania : 1
Proba dodania : 2
Added Dirt to inventory slot 4
Mining...
Mined Dirt 1!
Placed Stone at (-15, -9)
You are standing on Stone 2
Inventory Options
Show Inventory
Hit
Mine
Inspect
Place Block
Change Item
Remove from stack


Game
Niedz. 2.06 15:52
Game
Added Sword to inventory slot 1
Proba dodania : 0
Added Pickaxe to inventory slot 2
Mining...
Mined Stone 1!
Proba dodania : 0
Proba dodania : 1
Added Stone to inventory slot 3
Mining...
Mined Stone 1!
Mining...
Mined Stone 1!
Mining...
Mined Dirt 1!
Proba dodania : 0
Proba dodania : 1
Proba dodania : 2
Added Dirt to inventory slot 4
Mining...
Mined Dirt 1!
Placed Stone at (-15, -9)
You are standing on Stone 2
Inventory Options
Show Inventory
Hit
Mine
Inspect
Place Block
Change Item
Remove from stack


Game
Niedz. 2.06 15:51
Game
Added Sword to inventory slot 1
Proba dodania : 0
Added Pickaxe to inventory slot 2
Mining...
Mined Stone 1!
Proba dodania : 0
Proba dodania : 1
Added Stone to inventory slot 3
Mining...
Mined Stone 1!
Mining...
Mined Stone 1!
Mining...
Mined Dirt 1!
Proba dodania : 0
Proba dodania : 1
Proba dodania : 2
Added Dirt to inventory slot 4
Mining...
Mined Dirt 1!
Placed Stone at (-15, -9)
Inventory Options
Show Inventory
Hit
Mine
Inspect
Place Block
Change Item
Remove from stack