Projekt: Helden

Das Ziel des Spieles

Das Ziel ist es, stark genug zu werden, um den Drachen am Ende des Spiels zu besiegen und währenddessen zu vermeiden zu verlieren. Das Spiel wird so lange gespielt, bis entweder der Drache von den Helden besiegt wird oder bis alle Helden nicht mehr am Kampf teilnehmen können, sei es, dass alle Helden besiegt wurden oder, dass die, die noch leben, gerade bei dem Heiler sind.

Der Ablauf des Spieles

Der Spieler wird am Anfang gefragt, in welchen Bereich er gehen will. Er kann sich entscheiden in den erste, den zweiten, den dritten und den Boss-Bereich zu gehen. Zu Anfangs sollte er sich entscheiden in den ersten Bereich zu gehen, da die Helden noch nicht stark genug für den zweiten Bereich sind. Sie werden stärker, indem sie neue Waffen aufheben, die bessere Kampfwerte haben.

Wenn ein Held stirbt, wird dieser direkt zum Heiler geschickt und muss dort vier Kämpfe verweilen, bis er vom Heiler wiederbelebt wir. Wenn ein Gegner besiegt wird, lässt er eine neue Waffe fallen, die der Held aufheben kann und nach dem Beginn eines neuen Kampfes, oder wenn seine momentane Waffe kaputt geht, ausrüsten kann.

Nach einem Kampf wird der Spieler gefragt, ob er weiter in dem Bereich, in dem er sich gerade befindet, kämpfen will oder nicht, wenn er sich fürs Kämpfen entscheidet, wird ein neuer Kampf gestartet. Wenn er sich fürs nein entscheidet, wird er gefragt, ob er zum Heiler gehen will und danach, ob er zum Schmied gehen will. Bei dem Heiler kann der Spieler seine Helden einen Kampf warten lassen, nachdem der erste Held auf der Wartebank geheilt wird. Bei dem Schmied kann man seine Waffen für einen Geldbetrag reparieren lassen.

Bei dem Beschreiten eines höheren Bereiches kämpfen die Helden gegen stärkere Gegner, die aber immer bessere Waffen fallenlassen können.

In einem Kampf wird die Angriffsreinfolge durch die Geschwindigkeit der Gegner und der Waffen, die die Helden momentan tragen, entschieden.

Methoden und Klassen

Klasse Main

Methoden

Main()

Der Einstiegspunkt des Programms. Erstellt Instanzen von Healer, Schmied, Alle_waffen und waffen. Erstellt Instanzen von vier Helden (Paladin, Thief, Mage, Babar) und setzt ihre Anfangsstatistiken. Erstellt eine Instanz von Heldeninventar und fügt eine Standardwaffe (Fist) hinzu. Erstellt Instanzen von Alle_monster und einem Drachen. Erstellt eine Instanz von Gameplay und startet das Spiel.

Klasse Wuerfel

Die Klasse Wuerfel repräsentiert einen Würfel mit einer bestimmten Anzahl von Seiten.

Attribute: anzahlSeiten: private Variable, die die Anzahl der Seiten des Würfels speichert.

Parameter: anzahlSeiten: Die Anzahl der Seiten, die der Würfel haben soll.

Methoden

1: public int wuerfeln1(double anzahlSeiten)

Diese Methode simuliert das Werfen eines Würfels und gibt eine zufällige ganze Zahl zwischen 1 und der Anzahl der Seiten des Würfels (inklusive) zurück. Sie wird beim Fallenlassen von Waffen benutzt, indem sie Wahrscheinlichkeiten simuliert.

Parameter: anzahlSeiten: Die Anzahl der Seiten des Würfels.

Rückgabewert: Eine zufällige ganze Zahl zwischen 1 und der Anzahl der Seiten des Würfels.

2: public double wuerfeln2(double anzahlSeiten)

Diese Methode simuliert das Werfen eines Würfels und gibt eine zufällige Gleitkommazahl zwischen 1 und der Anzahl der Seiten des Würfels (inklusive) zurück. Die Gleitkommazahl wird auf zwei Dezimalstellen gerundet. Sie wird bei dem Berechnen des Matk von Helden sowie der Monster als Crit-Wert benutzt.

Parameter: anzahlSeiten: Die Anzahl der Seiten des Würfels.

Rückgabewert: Eine zufällige Gleitkommazahl zwischen 1 und der Anzahl der Seiten des Würfels (gerundet auf zwei Dezimalstellen).

Klasse Waffen

Die Klasse Waffen repräsentiert verschiedene Arten von Waffen mit unterschiedlichen Eigenschaften.

Attribute: name, Typ, Material, Magie, Smulti, speed, durability, used

Konstruktor: public Waffen(String name, String Typ, String Material, double Magie, double Smulti, int speed, int durability, boolean used)

Parameter: name: Der Name der Waffe. Typ: Der Typ der Waffe. Material: Das Material der Waffe. Magie: Die magischen Eigenschaften der Waffe. Smulti: Der Schadensmultiplikator der Waffe. speed: Die Geschwindigkeit der Waffe. durability: Die Haltbarkeit der Waffe. used: ob die Waffe schon benutzt wird.

Methoden

1: public double Stats(Waffen curent)

Diese Methode setzt die statistischen Eigenschaften der Waffe basierend auf ihrem Typ und Material.

Parameter: curent: Die aktuelle Waffe.

Rückgabewert: Im Moment wird immer 0.0 zurückgegeben.

2: public String getName(Waffen curent)

Diese Methode gibt den Namen der Waffe zurück.

Parameter: curent: Die aktuelle Waffe.

Rückgabewert: Der Name der Waffe.

Klasse Schmied

Die Klasse Schmied repräsentiert einen Schmied, der die Fähigkeit hat, Waffen zu reparieren.

Methoden

public void repair(Waffen waffe, Heldeninventar HI)

Diese Methode ermöglicht dem Schmied, eine Waffe zu reparieren, wenn der Held genügend Geld hat (100 Einheiten). Sie wird ausgeführt, wenn der spieler nach einem Kampf seine Waffen beim Schmied reparieren will.

Parameter: waffe: Die zu reparierende Waffe. HI: Das Inventar des Helden, das auch das Geld des Helden enthält.

Klasse Monster

Die Klasse Monster repräsentiert verschiedene Arten von Monstern im Spiel.

Attribute: name, typ, staerke, lebenspunkte, speed, verteidigung, Matk, Mdef, wmulti, tod

Konstruktor: public Monster(String name, String typ, int staerke, int lebenspunkte, int speed, int verteidigung, int Matk, int Mdef, int wmulti, boolean tod)

Parameter: name: Der Name des Monsters. typ: Der Typ des Monsters. staerke: Die Stärke des Monsters. lebenspunkte: Die Lebenspunkte des Monsters. speed: Die Geschwindigkeit des Monsters. verteidigung: Die Verteidigung des Monsters. Matk: Der Angriffsschaden des Monsters. Mdef: Die Verteidigung des Monsters während der Verteidigung. wmulti: Der Waffenmultiplikator des Monsters. tod: Ein Boolean, der angibt, ob das Monster tot ist oder nicht.

Methoden

1: public void stats(Monster monster)

Diese Methode setzt die statistischen Eigenschaften des Monsters basierend auf seinem Typ. Sie wird benutzt, wenn ein neues Monster initialisiert wird.

Parameter: monster: Das Monster, dessen Statistiken aktualisiert werden sollen.

2: public void angriff(Monster gegner)

Diese Methode simuliert einen Angriff des Monsters auf einen Helden. Der Angriffsschaden wird durch einen Würfelwurf mit einem kritischen Faktor bestimmt. Sie wird in der Methode ´gegenangriff´ verwendet.

Parameter: gegner: Der angegriffene Gegner.

3: public void verteidigen(Monster gegner)

Diese Methode simuliert die Verteidigung des Monsters gegen einen Angriff. Die Verteidigung wird durch einen Würfelwurf bestimmt.

Parameter: gegner: Der angreifende Gegner.

4: public int getVerteidigung()

Diese Methode gibt die Verteidigung des Monsters zurück.

Rückgabewert: Die Verteidigung des Monsters.

5: public int getLebenspunkte()

Diese Methode gibt die aktuellen Lebenspunkte des Monsters zurück.

Rückgabewert: Die Lebenspunkte des Monsters.

Klasse Heldeninventar

Die Klasse Heldeninventar verwaltet das Inventar der Helden, insbesondere deren Waffen und Geld, das durch das Besiegen von Monstern verdient wird.

Attribute: money: Die Menge an Geld im Inventar. Helden_waffen_inventar: Eine ArrayList, die die Waffen im Inventar der Helden enthält.

Methoden

1: public void inventar_ausgabe()

Gibt die Namen aller Waffen im Heldeninventar aus.

2: public String getwaffenName(int i, waffen waffe)

Gibt den Namen der Waffe an der angegebenen Position im Inventar zurück.

Parameter: i - Die Position der Waffe, waffe - Die Waffe.

Rückgabewert: Der Name der Waffe.

3: public void getMoney(monster Gegner)

Aktualisiert das Geld basierend auf dem besiegten Monster.

Parameter: Gegner - Das besiegte Monster.

Klasse Helden

Die Klasse Helden repräsentiert verschiedene Helden mit individuellen Eigenschaften.

Attribute: name: Der Name des Helden. typ: Der Typ des Helden (Paladin, Mage, Thief, Babar). staerke: Die Stärke des Helden. intelligence: Die Intelligenz des Helden. lebenspunkte: Die Lebenspunkte des Helden. verteidigung: Die Verteidigung des Helden. aktuelleWeapon: Die aktuell ausgerüstete Waffe des Helden. Matk: Der Angriffsschaden des Helden. Mdef: Die Verteidigung des Helden. nextw: Der Index der nächsten Waffe im Inventar. tod: Ein Flag, das angibt, ob der Held tot ist. fight_count: Die Anzahl der Kämpfe, die der Held beim Heiler verweilt hat. inWarteschlange: Ein boolean, der angibt, ob der Held in der Warteschlange ist.

Konstruktor: public Helden(String name, String typ, int staerke, int intelligence, int lebenspunkte, int verteidigung, waffen aktuelleWeapon, int Matk, int Mdef, int nextw, boolean tod, int fight_count, boolean inWarteschlange)

Methoden

1: public void stats(Helden Held)

Setzt die statistischen Eigenschaften des Helden basierend auf seinem Typ. Wird in Main verwendet.

2: public void angriff(Helden Held)

Berechnet den Angriffsschaden (Matk) basierend auf Stärke, aktueller Waffe, Intelligenz und crit.

3: public void verteidigen(Helden Held)

Berechnet die Verteidigung (Mdef) basierend auf der Verteidigungseigenschaft.

4: public void waffentausch(Helden Held, Heldeninventar HI)

Wechselt die Waffe des Helden basierend auf dem nächsten Waffenindex im Inventar und setzt die ausgewählte Waffe auf used = true.

Klasse Healer

Die Klasse Healer repräsentiert einen Heiler mit der Fähigkeit, Helden zu heilen und wiederzubeleben.

Attribute: RESET: Konstante für das Zurücksetzen der Farbe in der Konsole. GREEN: Konstante für die grüne Farbe in der Konsole. YELLOW: Konstante für die gelbe Farbe in der Konsole. wartebank: Eine ArrayList von Helden, die auf ihre Heilung warten.

Methoden

1: public void wartebank_schlange()

Überprüft, ob der erste Held in der Warteschlange zwei Kämpfe verweilt hat. Falls ja, werden die Statistiken aktualisiert, und eine Heilungsnachricht wird ausgegeben. Der Held wird dann aus der Warteschlange entfernt.

2: public void resurrect(Helden Held)

Überprüft, ob der übergebene Held vier Kämpfe verweilt hat. Falls ja, werden die Statistiken aktualisiert, und eine Wiederbelebungsnachricht wird ausgegeben.

Klasse Alle_waffen

Die Klasse Alle_waffen repräsentiert eine Sammlung aller Waffen im Spiel, mit der Möglichkeit, neue Waffen basierend auf bestimmten Kriterien zu erstellen.

Attribute: i: Eine Zählvariable, die den Index für die Position der neuen Waffe in der ArrayList alle_waffen angibt. alle_waffen: Eine ArrayList, die alle erstellten Waffen enthält.

Konstruktor: public Alle_waffen(int i)

Der Konstruktor initialisiert die Zählvariable i mit dem übergebenen Wert.

Methoden

public void newWaffe(int wmulti)

Erstellt eine neue Waffe basierend auf dem übergebenen Waffenmultiplikator (wmulti).

Der Typ der Waffe (Typ) wird durch einen Zufallswurf ausgewählt (Dolch, Schwert, Axt oder Stab).

Das Material der Waffe (Material) wird ebenfalls durch Zufallswürfe basierend auf dem Waffenmultiplikator bestimmt (Holz, Eisen, Dragonite).

Fügt die neu erstellte Waffe der ArrayList alle_waffen hinzu und ruft die Stats-Methode auf, um die statistischen Eigenschaften der Waffe zu setzen.

Klasse Alle_monster

Die Klasse Alle_monster repräsentiert eine Sammlung aller Monster im Spiel, mit der Möglichkeit, neue Monster basierend auf dem aktuellen Spielbereich zu erstellen.

Attribute: i: Eine Zählvariable, die den Index für die Position des neuen Monsters in der ArrayList alle_monster angibt. alle_monster: Eine ArrayList, die alle erstellten Monster enthält.

Konstruktor: public Alle_monster(int i)

Der Konstruktor initialisiert die Zählvariable i mit dem übergebenen Wert.

Methoden

public void newMonster(String Area)

Erstellt ein neues Monster basierend auf dem übergebenen Spielbereich (Area).

Der Typ des Monsters (typ) wird durch Zufallswürfe (wuerfeln1()) basierend auf dem Spielbereich ausgewählt. Fügt das neu erstellte Monster der ArrayList alle_monster hinzu und ruft die stats-Methode auf, um die statistischen Eigenschaften des Monsters zu setzen.

Klasse kampf

Attribute: MTod: boolean-Wert zur Verfolgung von Monster-Toden. HTod: boolean-Wert zur Verfolgung von Helden-Toden. i: Index-Variable. t: Index-Variable für die Auswahl des Ziels. alleHelden_tod: boolean-Flage für alle toten Helden. alleMonster_tod: boolean-Flage für alle toten Monster. c1: Instanz der Klasse wuerfel für die Generierung von Zufallszahlen. s: Instanz der Klasse Scanner für Benutzereingaben. ANSI Escape-Codes für die Farbgestaltung von Konsolentexten.

Methoden

1: public void Fight(Helden Held1,Helden Held2,Helden Held3,Helden Held4, monster Gegner1,monster Gegner2,monster Gegner3,monster Gegner4, Heldeninventar HI, Healer Healer, Alle_waffen AW)

Verwaltet die Hauptkampfsequenz. Behandelt Heldenzüge, Benutzereingaben und Sieg/Niederlage-Bedingungen. Ruft andere Methoden für Waffenwechsel, Angriffe und Überprüfungen auf. Erweckt tote Helden wieder zum Leben und verarbeitet die Wiederbelebung des Heilers. Sie wird in ´Area_Game ´ verwendet.

Parameter: Helden (Helden-Instanzen). Monster (monster-Instanzen). Heldeninventar (Heldeninventar-Instanz). Heiler (Healer-Instanz). Waffen (Alle_waffen-Instanz).

2: public void Heldattack(Helden Held, monster Gegner, int t, Alle_waffen AW, Heldeninventar HI)

Verwaltet die Angriffslogik, wenn ein Held ein Monster angreift. Bestimmt den verursachten Schaden, aktualisiert die Monster-HP und behandelt besiegte Monster. Verwaltet die Haltbarkeit der Waffe und fügt fallengelassene Waffen zum Inventar hinzu.

Parameter: Angreifender Held (Helden-Instanz). Zielmonster (monster-Instanz). Zielindex(t).

3: public void waffentausch_und_Ausgabe (Heldeninventar HI, Helden Held)

Behandelt den Prozess des Waffenwechsels für einen Helden. Sie zeigt das Inventar an und fordert den Benutzer zur Auswahl einer Waffe auf.

Parameter: Heldeninventar (Heldeninventar-Instanz). Angreifender Held (Helden-Instanz).

4: public void waffencheck (Helden Held, Heldeninventar HI)

Überprüft, ob die aktuelle Waffe des Helden kaputt ist. Falls ja, fordert sie den Benutzer auf, die Waffe zu wechseln.

Parameter: Angreifender Held (Helden-Instanz). Heldeninventar (Heldeninventar-Instanz).

5: Gegnerangriff (monster Gegner, Helden Held)

Verwaltet die Angriffslogik, wenn ein Monster einen Helden angreift. Bestimmt den verursachten Schaden, aktualisiert die Helden-HP und behandelt besiegte Helden.

Parameter: Angreifendes Monster (monster-Instanz). Zielheld (Helden-Instanz).

Klasse Gameplay

Attribute: alleHelden_tod: Ein boolean-Flage, das den Status aller Helden im Spiel verfolgt. Area: Ein String, der die aktuelle Spielregion darstellt.

Methoden

1: Game(Alle_monster Am,monster Dragon,Helden Held1,Helden Held2,Helden Held3,Helden Held4,Heldeninventar HI,Healer Healer, Alle_waffen AW, Schmied Andre)

Verwaltet den Hauptspielablauf. Koordiniert den Kampf gegen den Drachen und die Interaktion mit dem Heiler und dem Schmied. Überwacht Sieg-und Niederlagebedingungen.

Parameter: Am: Eine Instanz der Klasse Alle_monster für das Management von Monstern. Dragon: Eine Instanz der Klasse monster als der Boss. Held1, Held2, Held3, Held4: Instanzen der Klasse Helden. HI: Eine Instanz der Klasse Heldeninventar. Healer: Eine Instanz der Klasse Healer. AW: Eine Instanz der Klasse Alle_waffen. Andre: Eine Instanz der Klasse Schmied.

2: Area_Game(monster Dragon,Alle_monster Am,Helden Held1,Helden Held2,Helden Held3,Helden Held4,Heldeninventar HI,Healer Healer, Alle_waffen AW)

Verwaltet den Spielbereichswechsel. Koordiniert den Kampf in einem ausgewählten Bereich und ermöglicht die Interaktion mit dem Heiler und dem Schmied.

Parameter: Dragon: Eine Instanz der Klasse monster als der Boss. Am: Eine Instanz der Klasse Alle_monster für das Management von Monstern. Held1, Held2, Held3, Held4: Instanzen der Klasse Helden. HI: Eine Instanz der Klasse Heldeninventar. Healer: Eine Instanz der Klasse Healer. AW: Eine Instanz der Klasse Alle_waffen.

Begründung der jeweiligen Umsetzung der Klassen und Methoden

Die Umsetzung hat sich so ergeben, durch die Reinfolge, in der die unterschiedlichen Klassen erstellt wurden und das Verständnis, das ich während des Programmierens erlangt habe. Die Umsetzung der Klasen und Methoden hatte kein Fokus auf Effizienz oder Effektivität, sondern diente alleinig meinem Verständnis der Sprache und dem Spaß an dem Projekt. Es wurden einige irrelevante Methoden und Attribute dem Spiel hinzugefügt, da sie das Spiel interessant machen.

Probleme bei der Umsetzung

Hauptsächlich war der Syntax eine Hürde, über welche ich häufiger gestolpert bin. Zudem kamen Flüchtigkeitsfehler, die einige Zeit verschwändet haben. Rundum gab es aber noch keine Fehler, die nicht entfernt wurden oder werden können.