ReEvolution

Travail personnel

Ruppert Romeo
Tuteur: Tom Goedert

Inhaltsverzeichnis

Einleitung 1
Ziel und Ablauf meines Spiels 2
Unity 2
Programmiersprachen 3
Was ist C++ 3
Was ist C# 3
Algorithmen 4
Pathfinding 4
Pathfinding optimieren 6
Basic C# Syntax 7
Basische Operationen und Deklarationen 7
Funktionen 7
State Machine 8
Screenshots von meinem Spiel 9
Schlussfolgerung 11
Quellen 11

Einleitung

In den letzten Monaten habe ich mich mit einer neuen Programmiersprache, C# auseinanderzusetzen, und habe mich dann entschieden mein Spiel ReEvolution mithilfe von Unity zu programmieren.
Ich wurde mit meinem Spiel nicht fertig, jedoch habe ich mir Mühe gegeben, mein Spiel modular aufzubauen, so dass man einfach mehr Inhalt hinzufügen kann.
Hier schreibe ich über die verschiedenen Systeme, die ich in meinem Spiel verwendet habe und über die Basis von C#, jedoch werde ich mehr über mein Spiel schreiben.
Mein Spiel kann man hier für Windows herunterladen: https://github.com/RoStar2006/ReEvolution
~15.04.2021

Ziel und Ablauf meines Spiels

Das Ziel meines Spiels ist einfach gesagt das Überleben in der Natur.
Man Arbeiter steuern kann die Ressourcen suchen oder auch verarbeiten können.
Diese Arbeiter können gesteuert werden oder auch automatisch bestimmte Aktionen ausführen, wie das Sammeln von Holz oder das Transportieren von Ressourcen zwischen Gebäude, mit denen Ressourcen kann man dann neue Gebäude und Ressourcen herstellen.
Die Herstellung von neuen Ressourcen erlauben Verteidigungstürme Gegner fernzuhalten, jedoch sind die Angreifer einfache Waldbewohner, wie Wölfe und Wildschweine.
Diese Tiere können vereinzelt oder auch als Gruppe wandern und angreifen.
Jedoch musst du aufpassen, weil sie sich auch weiterentwickeln und dafür sorgen, dass du immer auf der Hut sein muss.

Unity

Ich werde jetzt über das Programm reden was ich verwendet habe, um mein Spiel zu erstellen.

Unity ist ein Computer Programm, das entwickelt wurde, um Spiele einfach und intuitiv zu erstellen, daher ist Unity eine „Game-Engine“, übersetzt Spiel-Engine.
In Unity kann man „AAA“ Spiele[1] oder auch Indie-Games[2] erstellen, jedoch werden meisten „AAA“ Spiele in alternativen Game-Engines programmiert, weil Unity die Programmiersprache C# verwendet und nicht C++.
In Unity kann man Spiele in 2D oder auch 3D erstellen, wenn nötig kann man auch zusätzliche Bibliotheken installieren.
Diese Bibliotheken fügen meistens etwas hinzu was dazu führt, dass das Spiel flüssiger läuft, oder auch neue Funktionen hinzufügt.
Zum Beispiel gibt es das DOTS Packet, das viele Bibliotheken beinhaltet, die einfaches Multithreading[3] erlauben, jedoch muss es nicht neue Möglichkeiten in der Programmierung sein.
Es kann auch etwas komplett Neues sein, wie das „neue Inputsystem“, die die Tastenbelegung einfacher soll machen.
Quelle 6, 7, 8)

[1] AAA Spiele sind Videospiele die von großen Verläge produziert und veröffentlicht werden (EA, Activision, Ubisoft)
[2] Ein Indie-Game ist ein Spiel, das von einer Person oder von einer kleinen Gruppe programmiert wurde.
[3] Multithreading ist, wenn dein Prozessor die Arbeit auf ander Kerne verteilt, die dann parallel arbeiten.

Programmiersprachen

Hier rede ich über die Programmiersprachen C++ und C# und ein bisschen über deren Unterschiede, jedoch ist es nicht wichtig die Details zu kennen.

Was ist C++

C++ ist eine Programmiersprache die zwischen 1979 - 1983 entwickelt wurde und mit dem ISO-Standard 1998 genormt.
Die kompilierte Form von C++ stammt hauptsächlich von C, jedoch sind sie sich nicht ähnlich, weil C++ hat, objektorientierte[4] Eigenschaften.
Die objektorientierten Eigenschaften von C++ sind nicht notwendig, weil man könnte C++ Code auch rein imperativ[5] schreiben, weil dann würde ein C-Compiler den Code auch akzeptieren und kompilieren.
Anzumerken ist, dass ein C++ Compiler auch problemlos C Code kann kompilieren.
Quelle 3)

Was ist C#

C# ist eine Programmiersprache, die in den 2000er von Microsoft, basierend auf das .Net Framework[6] entwickelt wurde.
Diese Programmiersprache gilt zur Familie der C Programmiersprachen, jedoch besitzt sie mehr Ähnlichkeiten zu Java als zu C++.
C# wurde während der Entwicklung von Java beeinflusst, jedoch hatte C# auch Java beeinflusst, das sorgt für die Ähnlichkeiten.
Grundsätzlich ist C# eine einfachere zu erlernende Programmiersprache wie C++, weil das .Net Framework manche Aspekte vereinfacht.
Eins dieser Aspekte ist die Speicherverwaltung, z.B muss man bei C++ die Nutzung des Systemspeicher selbst managen, also Reservieren und Wieder Freigeben von Systemspeicher, hingegen macht das alles die automatische Speicherverwaltung oder Garbage Collector vom .Net Framework automatisch.
Jedoch ist C++ eine effizientere Sprache als C#, weil es näher am Maschinen Code dran ist als C#, was auf schnellere Exekution von Befehlen hindeutet.
Quelle 4, 5)

[4] Objektbasiert: Das bedeutet, dass es sich um Objekte handelt, die Daten und Funktionen speichern
Objektorientiert: Das ist eine Erweiterung der objektbasierten Konzepte
[5] Programm das Anweisungen in einer Reihenfolge ausführt.
[6] Ein Framework bietet alle nötigen Bibliotheken, Komponenten, Laufzeitumgebungen und die Designgrundstruktur für die Entwicklung zur Verfügung.

Algorithmen

Eine kleine Einleitung zu Algorithmen kann nicht schaden, besonders wenn das nächste Thema über einen etwas „Komplexeren“ redet, daher möchte ich über die Definition eines Algorithmus reden.

Ein Algorithmus ist eine Reihe von Befehlen oder Regeln, die dann zu einem gewünschten Resultat führen, wie z.B., der Rubiks Cube, um den zu lösen ist sind eine Reihe von Algorithmen notwendig.
Algorithmen können sehr einfach oder sehr komplex sein, jedoch haben alle etwas gemeinsam, und zwar wenn man ihn immer dieselben Inputs[7] gibt, dann kommt immer wieder das gleiche Resultat.
Quelle 9, 10)

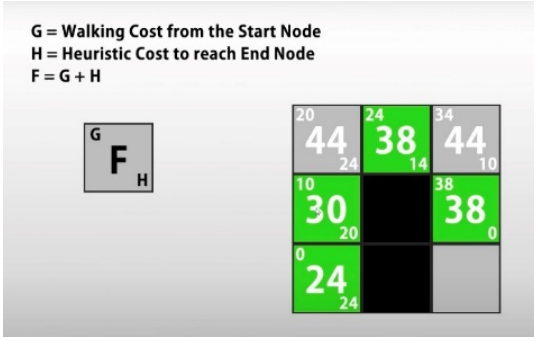
Pathfinding

Pathfinding ist einer meiner Hauptsysteme in meinem Spiel, daher es sorgt, dass sich alles Lebende bewegen kann, jedoch nicht irgendwie, sonders „schlau“.

Pathfinding bedeutet im deutschen „Weg suchen“, es gibt verschiedene Algorithmen, jeder Algorithmus sucht sein Weg auf eine andere Art und Weise. Ich habe den AStar (A*) Algorithmus herausgesucht, weil er schnell ist und trotzdem noch den kürzesten Weg findet.
Der A* Algorithmus könnte man als Erweiterung des Dijkstras Algorithmus sehen, jedoch erzielt der A* Algorithmus bessere Leistung und das, weil es ein Informierter[8] such Algorithmus ist.
Der A* Algorithmus benötigt einem Grid (Gitter), dieser Gitter hat eine fest eingestellte Größe, das Grid wird verwendet, um zu markieren, wo man drauf laufen kann und wo nicht.
Der A* Algorithmus benötigt jetzt nur noch eine Start Position und eine End Position, die Start und End Positionen müssen beide betretbar sein, also es darf kein Objekt auf diesen Positionen den Weg versperren.
Der Weg wird mit Hilfe von 5 Variablen berechnet, die G-Cost, H-Cost, den F wert und die Kosten für eine Diagonale oder Vertikale/Horizontale Bewegung, wenn man eine Diagonale Bewegung macht, werden die Kosten um 14 erhöht, jedoch bei einer Vertikale/Horizontale Bewegung werden die Kosten nur um 10 erhöht.

G-Cost: Dieser Wert zeigt die Kosten um von dem Aktuellen Feld bis zum Start.
H-Cost: Heuristische[9] kosten, um die end Position zu erreichen.
F: Der F Wert ist der Wert, wenn man die G-Cost und H-Cost addiert.

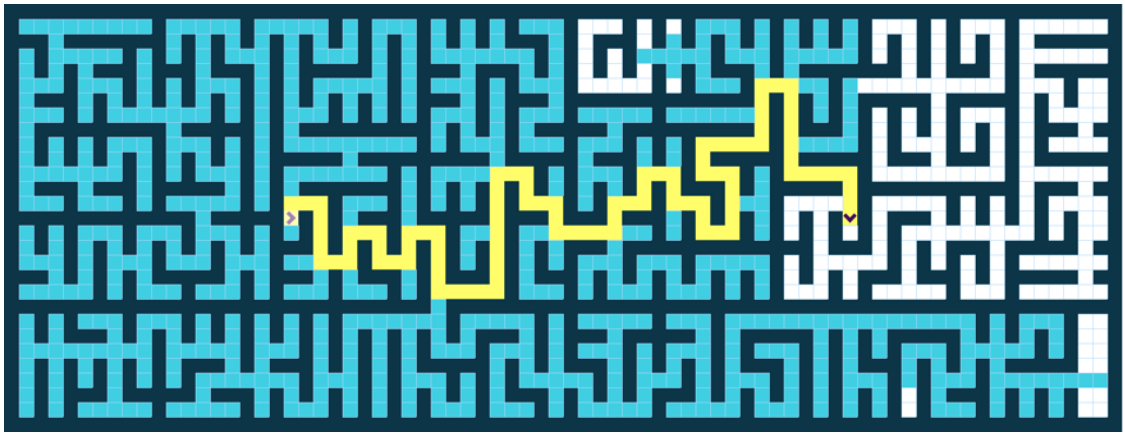
Der Algorithmus merkt sich jedes besuchte Feld auf dem Grid, dieses Feld nennt man dann eine Node, bereits besuchten Nodes werden in einer Liste abgespeichert.
Wenn der Algorithmus alle Nodes abgetastet hat und keinen Weg gefunden hat, dann bedeutet es, dass es keinen Weg gibt. Wenn der Algorithmus auf ein Feld mit dem H Wert 0 trifft, bedeutet es, dass diese Node die End Position ist und dass einen Weg gibt.
Der Algorithmus schaut jetzt in der Liste von bereits besuchten Nodes nach denen mit den niedrigsten F Wert, um somit dann den endgültigen Weg zu finden.
Quelle 1, 2, 13)



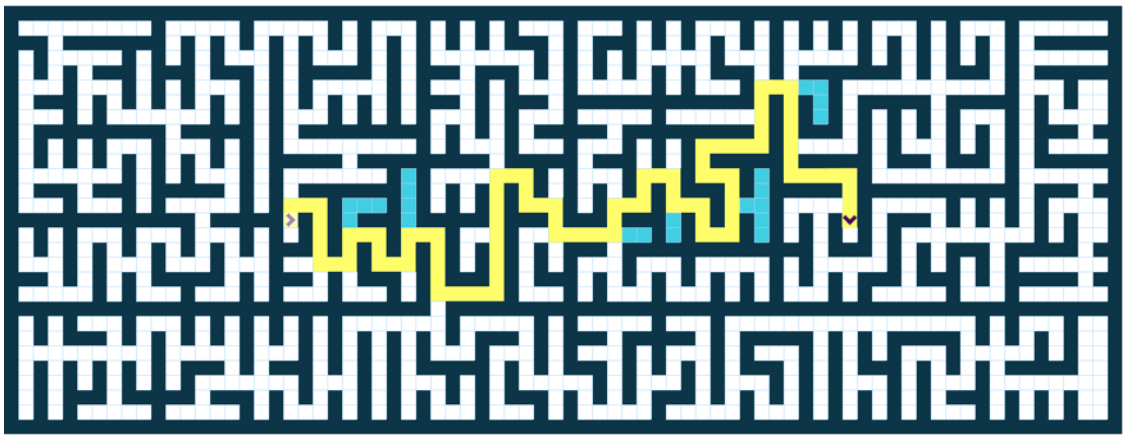
[7] Eingang
[8] Ein Informierter Sucher verwendet Heuristiken um die Berechnungen schneller auszuführen
[9] Eine Heuristik ist im Pathfinding eine Einschätzung wie weit entfernt ein Feld von der end Position entfernt ist

Hier sind 3 Beispiele von Pathfinding Algorithmen, einmal A*, dann Dijkstras Algorithmus und der Greedy Best-first Search Algorithmus. A* und Greedy Best-first Search sind Informierte Algorithmen, jedoch nicht der Dijkstras Algorithmus. A* und der Dijkstras Algorithmus haben eine 100% Chance, um den kürzesten Weg zu finden, jedoch nicht der Greedy Best-first Search Algorithmus. Das liegt daran das der Dijkstras Algorithmus „sogfältiger“ sucht, aber dann auch um ein vieles langsamer ist. A* besitzt daher Charakteristiken vom Dijkstras Algorithmus und Greedy Best-first Search Algorithmus, dies führt zu genauen und schnellen Berechnungen von Wege, jedoch hat er seine Schwachstelle bei großen komplexen Routen.

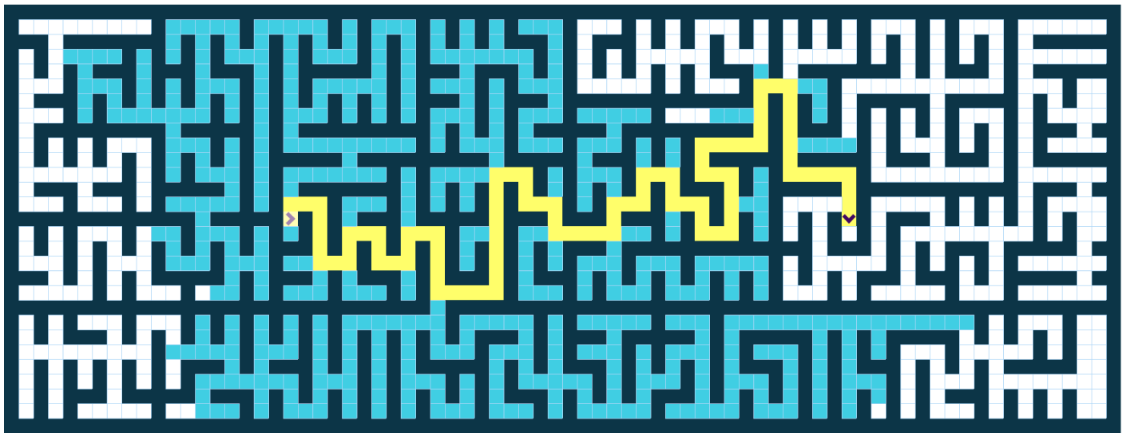
Dijkstras Algorithmus



Greedy Best-first Search Algorithmus



A* Algorithmus



Pathfinding optimieren

Weil das Pathfinding einer meiner Hauptbestandteile meines Spiels ist, muss ich es optimieren so dass es dem Spieler keine Ruckler gibt, die das Spiel Erlebnis zerstören würde.

Ich habe meine Optimierung damit begonnen, dass ich den Pathfinding Algorithmus auf einem anderen Thread des Prozessors laufen lassen, um so die Last gleichmäßig zu verteilen und um zu verhindern, dass das Spiel sich aufhängt, weil das Pathfinding zu lange braucht.
Um das Pathfinding auf andere Threads laufen zu lassen, musste ich neue Pakete/Bibliotheken installieren, die das Multithreading erlauben.
Ich habe mir das DOTS Packet heruntergeladen, dies enthält verschiedene Systeme, wie das Job System das sich um die anderen Threads des Prozessors kümmert um das Multithreading zu erlauben. Zusätzlich zum Jobs System gehört der Burst Compiler, was den endgültigen Code um ein vielfaches optimierter kompiliert, wie der übliche Compiler. Im DOTS Packet gibt es außerdem noch das Entity Component System (ECS) System, was das Jobs System nimmt und es benutzt, um die Logik von den einzelnen Objekten zu berechnen, jedoch kommt das mit dem Preis der Komplexität.
Alles kombiniert, erhält man das DOTS System, mit dem man sein Pathfinding oder anderes optimieren kann.

Unten habe ich die oben erwähnten Optimierungen verwendet und in einer Tabelle eingetragen, diese Tabelle habe ich in verschieden Spalten aufgeteilt.
Die Spalte „Versuch Nummer“ soll zeigen bei welchem Test man ist, also „Test 1, Test 2, 3…“.
Die Spalte „Job System“ soll mit Ja, Nein zeigen ob ich das Unity Job System verwendet habe, dasselbe gilt für den Burst Compiler und das Entity Component System (ECS).
Folgend kommt die Spalte „Anzahl an Berechnungen“ die zeigen soll wie viele Berechnungen liefen. Jedoch ist anzumerken, dass sie nicht alle gleichzeitig gestartet wurden, sondern wurden jede Sekunde neue Berechnungen hinzugefügt, bis ich die gewünschte Anzahl erreicht hatte.
Jede Berechnung hatte sich selbst wiederholt, wenn sie erledigt, wurde
Die letzte Spalte Dauer von Berechnungen in ms zeigt an, wie lange es dauerte, um die Anzahl an Berechnungen zu berechnen.

Versuch Nummer	Job System	Burst Compiler	ECS	Standard Objekte	Anzahl an Berechnungen	Dauer von Berechnungen in Millisekunden
1.	Nein	Nein	Nein	Ja	1	21.366
2.	Nein	Nein	Nein	Ja	10	/
3.	Ja	Nein	Nein	Ja	10	72
4.	Ja	Ja	Nein	Ja	10	2
5.	Ja	Ja	Nein	Ja	100	9,5
6.	Ja	Ja	Nein	Ja	1000	75
7.	Ja	Ja	Ja	Nein	10000	~3
8.	Ja	Ja	Ja	Nein	100000	~25
9.	Ja	Ja	Ja	Nein	200000	~60
10.	Ja	Ja	Ja	Ja	100	~9
11.	Ja	Ja	Ja	Ja	500	~30
12.	Ja	Ja	Ja	Ja	1000	~60-100

Quelle 3)

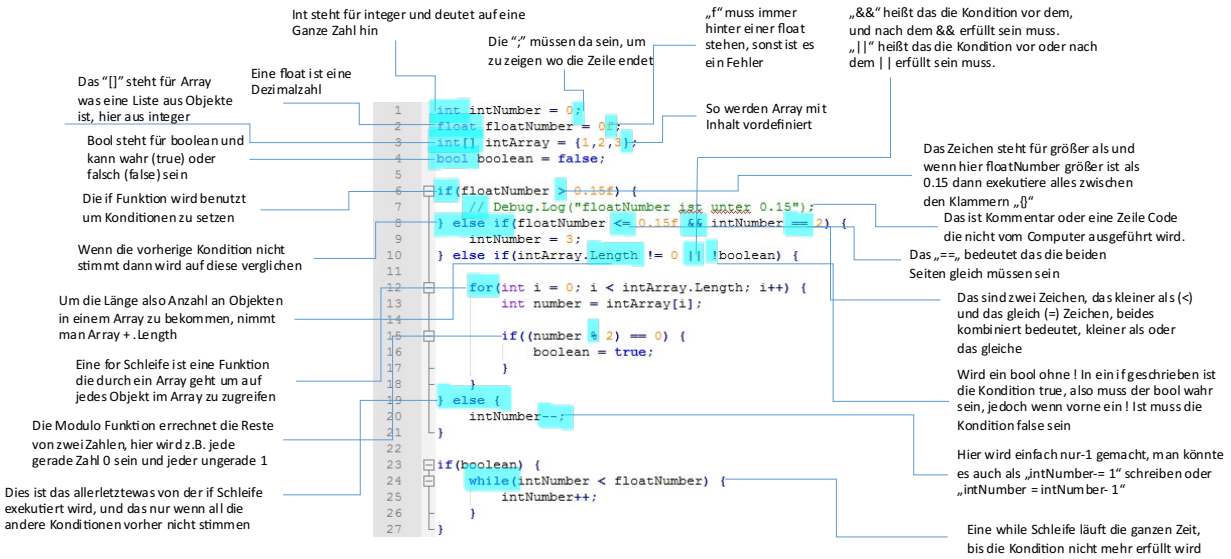
Basic C# Syntax

Hier erkläre ich die Basis für die Programmierung in C#, ich möchte es gerne erklären, um zu zeigen, dass das Programmieren eigentlich einfach ist und nur kompliziert aussieht.

*Die hier gezeigten Funktionen können von anderen Bibliotheken stammen (Unity Engine Bibliothek, ect…)

Basische Operationen und Deklarationen

Hier beschreibe ich verschiedene Basische Variable, Type und Methoden, angemerkt ist, dass es egal ist wie deine Variable heißt.



Funktionen

Hier habe ich versucht mit einem einfachen Beispiel eine Einleitung zu den Funktionen zu erstellen, es sei angemerkt, dass es egal ist, wie deine Funktion heißt.

State Machine

Hier möchte ich gerne um ein System reden was in meinem Spiel die Logik der Gegner und Arbeiter enthält und managed.Als ich dieses System entdeckt habe, habe ich meine Gegner und Arbeiter Logik komplet neugeschrieben, weil eine State Machine eine großen Überblick ermöglicht.

Eine State Machine ist eine Beschreibung in welchen Status sich die Maschine befindet, Idle[10] wird oft als Basis verwendet, dann kann man sein eigener Status einfügen wie unten z.B. Move für das Bewegen vom Spieler und Attack für das Angreifen von Feinden, somit kann ich Fehler vermeiden, wie das Angreifen während des Bewegens. Dar Status kann nur in einem sein, also Idle oder Move oder Attack, jedoch muss es ein von denen sein.

Hier kann man dann seine eigene Variablen einfügen, hier habe ich es so eingerichtet, dass die Variable State eins der drei „Positionen" hat.

Enum wird verwendetum ein eigenen Typ zu erstellen

Eine Variable mit dem Typ State erstellen

Die Variable state sicherheitshalber auf Idle setzen.

```
1  enum State = {
2      Idle,
3      Move,
4      Attack,
5  }
6  State state;
7
8  void Start() {
9      state = State.Idle;
10 }
11
12 void Update() {
13     switch (state)
14     {
15         default:
16         case State.Idle:
17             // Idle
18             break;
19         case State.Move:
20             // Move around
21             break;
22         case State.Attack:
23             // Attack enemy
24             break;
25     }
26 }
```

Eine switch Methode hat eine Variable als Input, die dann unten auf verschiedene Konditionen kontrolliert wird, jedoch wenn keine stimmt, wird die default Kondition ausgeführt
Die switch Methode hat Ähnlichkeiten zur if Methode

Case heißt einfach nur wenn die Input Variable mit der Kondition hier State.Idle übereinstimmt, dann exekutiere jede Zeile unter dem case

Break verwendet man um Loops (Schleifen) aufzulösen und zu stoppen.

Quelle 11)

[10] Idle ist ein Status, indem wenig oder nichts passiert, z.B. wird im Idle Status kein Code ausgeführt.

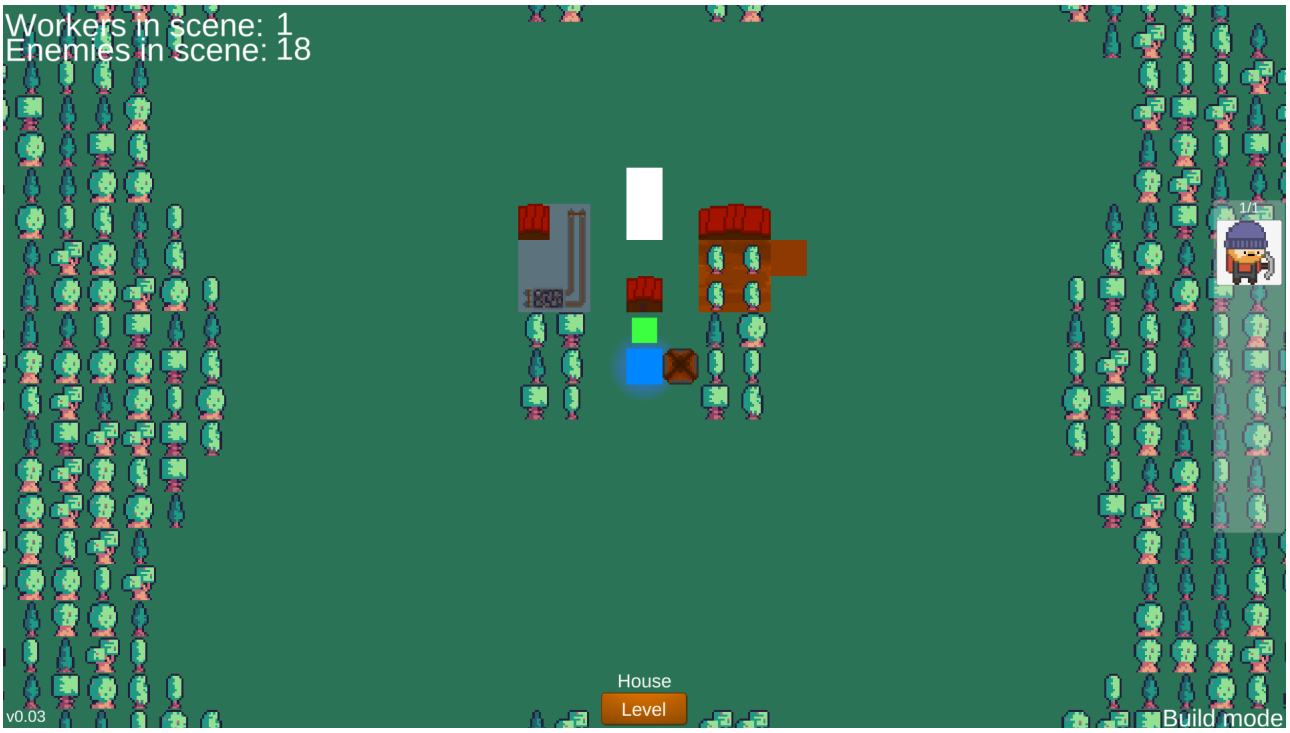
Screenshots von meinem Spiel

*Diese Screenshots können veraltet sein und nicht mit dem aktuellen Status des Spiels übereinstimmen.

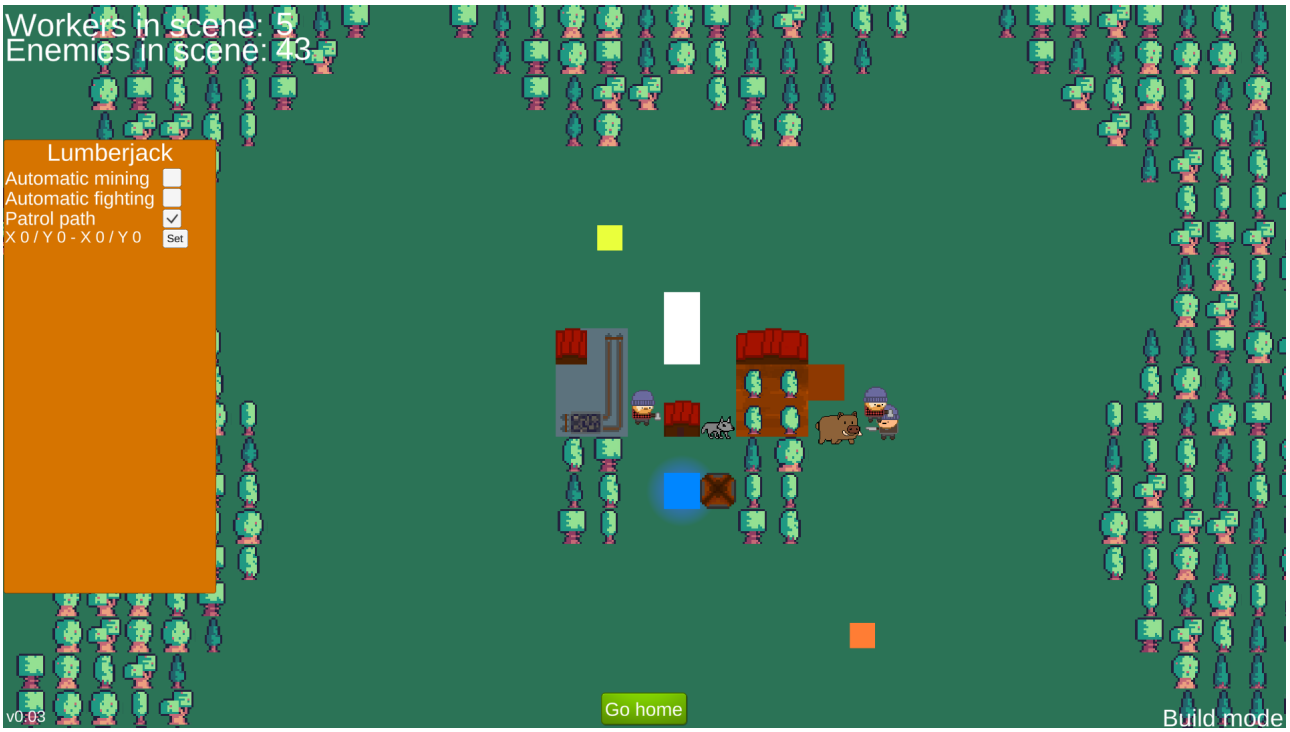
Hauptmenu meines Spiels

Pausenmenu im Spiel

Häuser die Arbeiter enthalten, die für Sammlung von Ressourcen und zur jagt, gebraucht werden können



Patrollieren von Punkt A zu Punkt B und das Angreifen von Gegner, sammeln von Ressourcen



Quelle 14 / Mein Spiel)

Schlussfolgerung

Mit diesem TraPe habe ich eine neue Programmiersprache erlernt und viel Erfahrung gesammelt. Ich hatte viele Probleme, auf die ich Lösungen finden musste. Jedoch mussten die Lösungen noch performant sein, damit das Spiel nicht „stottert“. Dieser Prozess vom Optimieren war anstrengend, jedoch hat es Spaß gemacht.

Mein Spiel macht mir Spass, jedoch ist es noch ein bisschen kompliziert zu spielen und zu erklären, darum plane ich in Zukunft das Spiel zu ändern, jedoch das Prinzip zu behalten.

Quellen

1) https://clementmihailescu.github.io/Pathfinding-Visualizer/#
2) https://youtu.be/1bO1FdEThnU
3) https://dev-insider.de/was-ist-c-a-581368
4) https://www.dev-insider.de/was-ist-c-a-846162/
5) https://www.youtube.com/watch?v=GRfQsj87wEE
6) https://unity.com/dots
7) https://unity.com/de
8) https://de.wikipedia.org/wiki/Unity_(Spiel-Engine)
9) https://medium.com/predict/what-is-an-ai-algorithm-aceeab80e7e3
10) https://praxistipps.chip.de/was-ist-ein-algorithmus-einfach-erklaert_41355
11) https://www.youtube.com/watch?v=db0KWYaWfeM
12) https://www.mygreatlearning.com/blog/best-first-search-bfs/
13) https://www.javatpoint.com/ai-informed-search-algorithms
14) https://github.com/RoStar2006/ReEvolution