Anton Rautanen

Designing and Implementing AI for a Car Game

Case Astalo Games

Liiketalous
2018

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Anton Rautanen
Opinnäytetyön nimi	Tekoälyn suunnitteleminen ja toteutus autopeliin
Vuosi	2018
Kieli	Englanti
Sivumäärä	29
Ohjaaja	Raija Tuomaala

Opinnäytetyön tarkoituksena on tutkia, suunnitella ja toteuttaa tekoäly autopeliin käyttäen Unity-pelimoottoria. Toimeksiantajana toimi Astalo Games Oy ja projektissa oli mukana toimeksiantajan johtava ohjelmoija.

Peleissä yksi tärkeä osa on tietokoneohjatut vastustajat eli tekoäly. Tässä työssä tutkitaan kahta eri tapaa luoda tekoäly autopeliin ja toteuttaa sen asiakkaan antamilla vaatimuksilla.

Työn aikana tarvitsin erilaisia tekniikoita, jotka helpottivat työtä, kuten tilakone, jolla saatiin erilaisia käyttäytymisiä tilanteisiin. Loin myös yhteisen rajapinnan, mikä laski koodin ylläpitoa ja testausta. Työn lopputuloksena sain toimivan tekoälyn, jota voidaan jatkokehittää eteenpäin asiakkaan toimesta, esimerkiksi parantamalla laskentoja, tilakonetta ja linjoja, mitä pitkin tekoäly kulkee.

Avainsanat tekoäly, peli, tilakone, Waypoint navigaatio systeemi

VAASA UNIVERSITY OF APPLIED SCIENCES
Bachelor of Business Administration in Information Technology

ABSTRACT

Author	Anton Rautanen
Title	Designing and Implementing AI for a Car Game
Year	2018
Language	English
Pages	29
Name of Supervisor	Raija Tuomaala

The object of this thesis was to study, design and implement AI for upcoming Astalo Games Oy car game using Unity game engine and C# programming language. This project was done with lead programmer of Astalo Games Oy.

One of the important feature in games is a computer controlled player that is artificial intelligence. In this thesis were studied two different ways to create AI for the car game and implement it with customers requirements.
During the work, it was noticed that we needed different techniques that would help the work, like finite state machine that enables us to create different behaviors on different situations, Common interface to AI and back, which lowered code upkeep and testing. The end production was working AI that the client could develop even further, for example improving the calculations, state machine and lines, which the AI uses to drive.

Keywords Artificial intelligence, video games, finite state machine, Waypoint navigation system

CONTENTS

TIIVISTELMÄ

ABSTRACT

Table of Contents

1 Introduction....7
2 AI in games....9
2.1 Rubber band....9
2.2 Waypoint navigation system....10
3 Finite-state machine....11
4 Unity- Game engine....13
4.1 Scripting languages....13
4.2 Unity navigation system....16
5 Requirements....18
6 Research Subjects....19
6.1 Standard asset example....19
6.2 Unity navigation system....19
6.3 Standard asset example limits....19
6.4 Navigation system limits....19
6.5 Pros and cons of standard asset example and unity navigation system....20
6.6 Conclusion....20
7 Implementing the ai....21
7.1 Designing the code....21
7.2 Implementing and testing....23
8 Conclusion....26
9 REFERENCES....28

Figures

Figure 1: Example for state diagram.....10

Figure 2: Default C# Class in Unity.....13

Figure 3: Unity's main event loop.....14

Figure 4: Racing state machine transitions.....21

Figure 5: Interface to send command to car from AI.....22

Figure 6: State machine design for AI.....23

Figure 7: 3 AI and one human player.....26

Table 1: Transition table made from Figure 1.....11

List of abbreviations

AI	Artificial intelligence
MonoBehavior	Base class for every unity script
Pathfinding	Plotting the shortest route between two points
Rubber band	Technique that changes speed of the AI
State machine	An abstract model of computation
Waypoint system	System that contains collection of points from developer can create paths
Standard asset	Collection of widely used assets

1 INTRODUCTION

Artificial intelligence (or AI in short) is one of the core features in modern video games from Overwatch to Binding of Isaac. AI is, in most cases a needed feature to make the game more interesting and challenging, while games without AI require a second human player to play. One of the first game with AI was Qwak (1974) which goal was to shoot ducks that came from the bottom of the monitor.

The research questions for this work are as follows: what kind of AI techniques are there? What different research subjects are there? What kind of AI can be created from a selected research subject?

In this work were introduced to the state machine, an abstract machine, a waypoint navigation system and a unity navigation system.

To limit this work, the focus is on two research subjects: Standard asset example and Unity navigation system.

This thesis is going to study what kind of artificial intelligence there is in video games, compare two of them and produce it for the client that is a local game studio called Astalo Games and their lead programmer. We also see what finite state machine is and what it can do for AI.

Astalo Game is a Vaasa based game company founded in 2016. The company consists of six persons and they have few past projects. Their current project is Street Heat, a four-player arcade racing game.(Astalo Games, 2018)

This thesis is constructed as follows: in the second chapter we look into AI in games, its history and two technique that are used regularly in games. Chapter three will explain what is finite state machine, how it is created and what helpful tools there are to create it. Chapter four introduces Unity game engine and its programming language c#. From chapter five to chapter seven we start to first design our AI from clients given requirements, then study two of the research subjects and after compering them, select one of them. Chapter seven look at the work and present the final product.

2 AI IN GAMES

AI and games have a long history together from the Alan Turing's algorithm for playing chess to modern video games like Counter Strike: Global Offensive, the AI is important part of the game. From the beginning AI's purpose was to test whether computers could solve tasks that would seems to require “as intelligence”, for example play chess against a human player. While this purpose is still the main point for AI, games have also risen to be excellent research target for AI's with games like the Starcraft game series and Dota 2. The StarCraft games have high complexity that is from the multi-objective tasks to controlling multiple and dissimilar units in a game world that offers little information. This complexity means that AI has difficulties to beat human players with average skill level in the game, so the AI has only achieved amateur level. (Yannakakis 2018, 8-9, 16-17) Modern games have many different technique and AI types, for example Rubber band technique and Waypoint Navigation system.

2.1 Rubber band

Some games might have basic AI that follows and attacks a players on sight, while others might have little more complex behavior. An example of this is a racing where AI tries to be on the lead and at the same time keep a specific distance to a player, to keep it challenging. This technique is called rubber band and its purpose is to prevent the player from getting too far ahead of the computer or too behind from it. To succeed in this, AI either enhance its ability like driving faster that it was designed for or by inhibiting the abilities of the player’s vehicle or both. (Gamasura 2018)

2.2 Waypoint navigation system

An other technique that is used in a racing game is Waypoint Navigation system. Waypoint Navigation system is a collection of points are which the AI uses to navigate in the game world. The points in a collection is either generated by an application or by a developer. If the points are generated by an application the path might be shorter and/or going the wrong way depending on the algorithm that created it, but the generating is much faster than by hand. If the collection is created by hand, it will take more time but the path will function as the developer designed it to.(Bourg, Seeman 2004, 170-174)

3 FINITE-STATE MACHINE

The finite-state machine, often referred as state machine is an abstract machine that can exist in one of several different and predefined states. The state machine can also define a set of conditions that determine the transition from state to state. The state machine allows the development of creating different behaviors to AI. For example theres a guard standing somewhere and by default he stands there looking if a player or other character comes in his field of view. When the player or other character comes near the guard, the guard start to chase them until either the player is fighting him or is too far away from him. This can be visualized with a state diagram as seen in Figure 1.

Figure 1: Example for state diagram

While the example shown in figure 1 is simple, real world machines would be much more complex and therefore creators need some way to represent the state machine and its transitions: enters the transition table. (Raganwald, 2018)

	Guard	Chase
Guard	-	Player is close
Chase	Lost player	-

Table 1: Transition table made from Figure 1

Transition table is a table that show what transitions it has in its current transition to the target state. Table 1 based on Figure 1 has only two (2) states and transitions. It tells us what transitions state “Guard” needs to proceed to state “Chace” and that is “Player is close” transition. From this we can create more easily code that resembles the state machine and add more states/transitions with ease. (Raganwald, 2018)

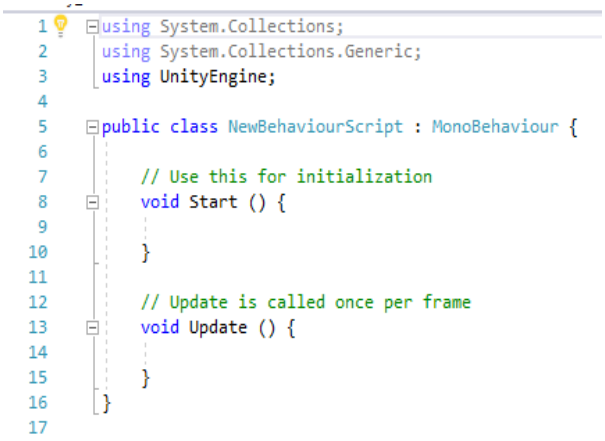
4 UNITY- GAME ENGINE

Unity is a multi-platform game engine developed by Unity Technologies. In Unity you can develop 2D-3D applications or games for PC, MacOS, Linux and modern consoles like PlayStation 4, Xbox One and Nintendo Switch.

4.1 Scripting languages

Unity has two scripting languages that developers can use to create their games: JavaScript and C#. C# is the most used scripting language used in Unity with 80.4% usage, while JavaScript had 18.9% in 2014 (Unity Technologies, 2014 a). In August 2017 Unity published an article about JavaScript and how they have conversion tool in development to transform JavaScript code to C#. They are slowly deprecating support for JavaScript so the Unity's developers can use more resources and focus only on C#. (Unity Technologies 2017 b). In this thesis C# is used as the scripting language for few reasons: the whole project is made using C# and it's currently the most supported scripting language, meaning there is plenty of documentation and tutorials to use.

When creating a new C# script in Unity, Unity adds default functions to it as seen in figure 2.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figure 2:Default C# Class in Unity

The MonoBehavior is a core for all game scripts. It contains reference to current game object and entity interface where you can query other scripts to be used in the current instance.

MonoBehavior also has functions that are overridden by the developer. Figure 3 shows these events and the order in which they are executed.

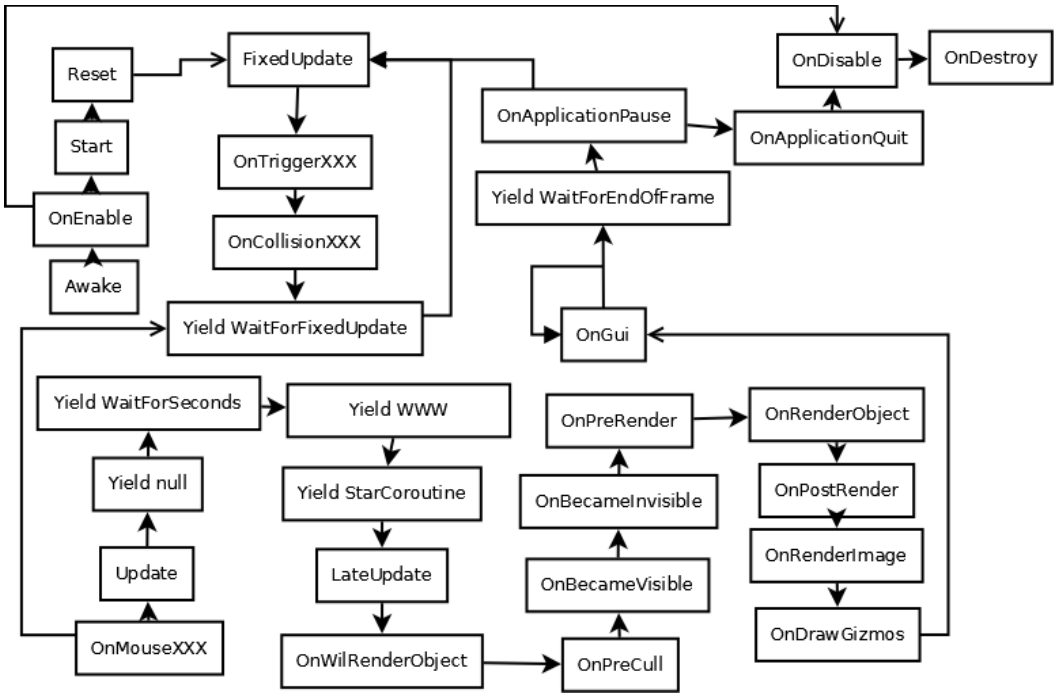


Figure 3: Unity's main event loop

Unity's many events can optionally be overridden by developers, for example to initialize current object with data in “start” event. Two of the events that are overridden by default is “Update” and “FixedUpdate”. These events are used in most cases to update the current object, it values and to end the game. In this work we are using following events are used: Awake, FixedUpdate, OnCollisionStay, OnCollisionEnter and OnDrawGizmo. “OnCollisionStay” and “OnCollisionEnter” are physics events that are called when a game object is colliding with a different object that has a collider component. A collider is component that defines the shape of an object for physical collision. These events can be used, for example to play sound when a car hits a wall. (Unity Technologies 2018 a)

“Awake” is an event that is called when an object is created and “FixedUpdate” is called every fixed frame that is by default every 0.02 seconds. “OnDrawGizmo” event is used when a developer whants to visualize an object. (Unity Technologies 2018 b)

4.2 Unity navigation system

Unity has a built-in navigation system that allows to create characters which can navigate in the game world. A navigation system gives information of the world to the character, so that it can reach the other side of an obstacle or walk to the second floor without hitting or getting stuck in corners or walls. Unity NavMesh contains the following pieces:

- NavMesh

  A data structure which describes the walkable surface and is created when setting the object to static mesh. This is created by baking the geometry to memory to be used on runtime.

- NavMesh Agent

  A component that helps create AI. NavMesh agent contains methods to move the AI and calculate optimal path while taking account objects in the world, like a box that blocks your path.

- Off-mesh link

  A component that enables connection between two separated meshes so that an agent can move between them for example, a door that you can open or jumping over an obstacle.

- NavMesh Obstacle

NavMesh Obstacle is an obstacle, which AI cannot go through, and therefor it must either be avoided or find alternative path. (Unity Technologies 2017 a)

Unity navigation system is a great system to create basic AI movement in a short time for genres that do not require fast moving objects like puzzle and shooter games, but for racing games it requires more work, like almost recreating the pathfinding system to produce looping circular paths.

5 REQUIREMENTS

At the start, requirements were somewhat simple from the client: create basic AI that follows the most optimal path from start to finish. The AI must also avoidin buildings, cars and other obstacles on this path. It must include an ability to be competitive against other drivers whether it is a human player or other AI.

The last requirement was that there will be adjustable parameters for AI, like maximum velocity, how aggressive AI can be towards other drivers and amount of error that the AI can make so that the client can change and create different levels of difficultys for the AI.

6 RESEARCH SUBJECTS

First to clarify, the word “standard asset” means the collection of widely used assets by Unity users. This chapter introduces to the chosen subjects.

6.1 Standard asset example

First the research subject was standard assets example of an AI car that uses waypoints to drive around the lap. The idea in this is that the car uses same core code to move forward that the player uses, but tries to follow a waypoint system to stay on the track.

6.2 Unity navigation system

The second research subject was Unity's own navigation system that uses navmesh to move in the game world. Navmesh enables to map paths faster and automate the moving with the cost of configuration.

6.3 Standard asset example limits

The limit that was discovered immediately when inspecting the example was the waypoint system itself. Speed and steering calculations are created for standard asset example, so they might not work the case car and require large modifications.

6.4 Navigation system limits

Navmesh calculates a path to move around in a scene and finds the shortest path to target, but does not take into account of laps and with this can't create looping path. The agent also requires more configurations to allow the car to drive smoothly in the race.

6.5 Pros and cons of standard asset example and unity navigation system

The standard asset example there has a few good sides and the same amount of negative sides. The positive sides where that it’s fairly easy to integrate and change to this project. Speed, gas, brake and steering calculations are easily replaced and waypoints are easily created and are changeable, even on runtime. The negative sides was that reversing the waypoints is difficult, even with a given tool that didn’t work as expected. the requires modification so that AI can take account of turns so it won't crash to buildings or other obstacles and the default calculations of steering, gas and brake didn’t work with the car so they needed to be changed.

The unity navigation system’s positive side was that it has an integrated system for path finding, and the scene is baked almost automatically, so an agent can move around the world without calculating the path every time. The only negative side was that there is now easy way to create that to follow and loop.

6.6 Conclusion

In the end the standard asset example was selected, because it gave more freedom to control the car's steering, brakes, gas and path that the navigation system didn't offer. Also the navigation system would have given much more work, like creating a waypoint system from scratch to easily create paths in each map.

7 IMPLEMENTING THE AI

7.1 Designing the code

In the last chapter one out of two research subjects is elected is going to be developed. The initial job was to design the connection from AI to car control. At first was direct control from AI code to car control, but that would require changes to AI asset every time the car behavior is changed. So, an interface was created that allowed easy upkeep with almost no changes to AI or car code when either is changed. When the connection from AI to car code was designed, control from AI code to track objects, like limitations, was also required so we designed interface for the AI called IAI. This interface would allow to set limits like how much gas and brake the AI can use, turn boost, set difficulty, reset current waypoint and change it to an other and get distance to the next waypoint.

While we implemented the interface and tested the current AI, we noticed that there is need to change the behavior of the AI in some specific situations. the state machine designed to allow more control over situations and gas/brake values.

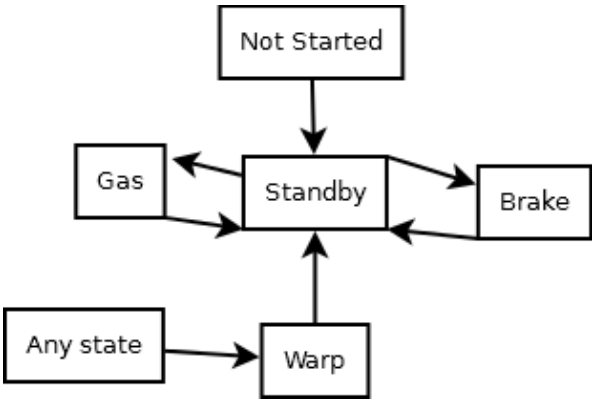


Figure 4: Racing state machine transitions

In Figure 4 you can see the initial layout for the state machine that would allow the behavior change on runtime. While the design of state machine was simple, it required even higher machine that allows changes to a different state machine, so we created the state machine shown in picture 6. We have also designed interface, which allow multiple state machines.

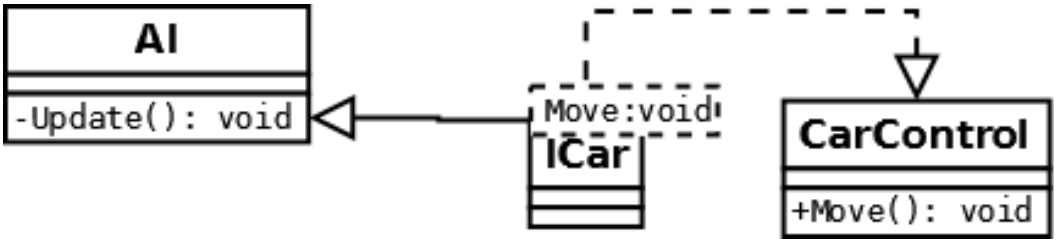


Figure 5: Interface to send command to car from AI

7.2 Implementing and testing

After the initial design was done, we started to integrate the standard asset to the current car code and at the same time we created the interface that shows in figure 5. First modifications to the code wore to remove references to standard asset classes that weren't use and implement IAI interface, so objects in a scene can change settings of the AI example. AI drives to a checkpoint and changes maximum usage of the brake, so the AI will only use the gas until the next checkpoint and changes the settings again. After modifications was done, I started to create first waypoint network in the first map, which included the possibility to create a shortcut for the AI. This meant that we needed to create flexible way to change or modify the waypoint network, so that AI could use this shortcut. The solution for a flexible waypoint was to copy the current waypoint and change it so that it go through the shortcut. But the whole system was abandoned, because there was no real need for shortcuts and changes to control points that would change settings of AI, like gas and brake usage, turning boost and max speed, by sending new values to AI code. With the control point we started to get faster lap times for the car, starting from 25 seconds to 14 seconds, while the best one was around 9-10 seconds with turn boost and max speed twice the original.

Figure 6 shows the final implemented state machine interface and two of its states.

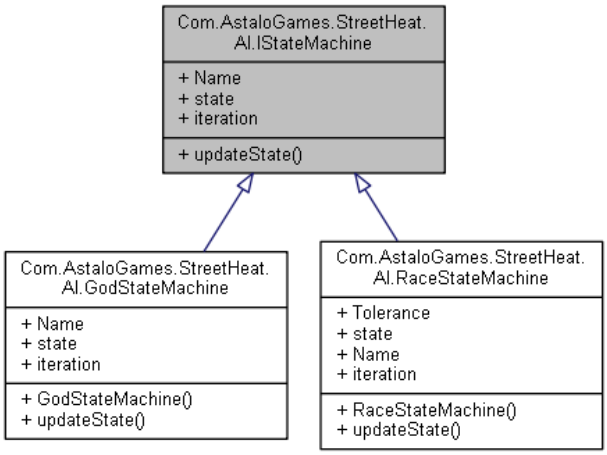


Figure 6: State machine design for AI

The first is RaceStateMachine, which is used when the car is driving in the game and the second is GodStateMachine, which is the upper level state machine that controls what state machine is in use in the game. There was a plans to implement a stuck state machine but it was canceled in favor of warp state, so the upper level machine only contains one lower machine. The Stuck state machine was collection the states that are used to get free from situations where can could not go forward. On the other hand, the warp state is just as state that teleports to the last checkpoint after being still for two seconds. The state machine work is somewhat simple: in every update it's checked if the current speed is equal or greater than the current limit. The state is changed to standby so it will try to stay in the limit, but if the current speed is passed the limit plus tje given tolerance, the state is changed from standby to brake, so that AI will slow to the limit.

When the AI got faster lap times in the first map, I started to create a waypoint network for the second map that included hard turns, changing height for the track, places where the car could fall or get stuck and the train. The workflow the was same as in the previous map: create optimal waypoint network for the AI and get as fast lap time as possible, however we also wanted to make AI look like a

human is controlling it like more swigging in the track, changing turn boost that make the car move more faster, and the max speed. While getting an optimal waypoint network, gas and brake calculations were looked into more, as they were causing over-usage of brake when there was no need to use brakes. This problem was solved by adding some tolerance in to the state machine transition, which is the max limit + 10% of the max limit.

The final task which was never tested or finished was the train avoidance state in the lower state machine. In some maps there might be train tracks that go across the road and there is a chance that trains ride using that track. The idea of this state was that the AI would check if there was a train coming in the tracks. If a train was coming the AI would stop and wait for the train to pass. It was never finished, because there was no real need for this feature, mostly because the car would be either too fast to get hit by the train, or too slow for this and it would be more fun to see the AI getting crushed by the train.

8 CONCLUSION

The goal of this thesis was to study, compare and create AI for the upcoming racing game, and it was completed with almost every given requirement done. While there were some big problems like the random usage of brakes when driving at max speed, the thesis was somewhat a success, and the work is now in a good state to evolve more.

The thesis centered on AI and the following questions: what kind of AI technique are there, what two different research subjects are there and what kind of AI can be created from the selected research subject. The navmesh was tested using already existing code, but it was not suitable for this work. On other hand, the standard asset example was perfect for this project, with its waypoint navigation system for creating the desired path for the used AI. This also meant that there was a need to create a state machine to have more control over the car behavior and get it free from stuck situations.

The given requirements were almost completed, except for the error free zones gave given some trouble from the beginning, but this was replaced by control points.

While the project is working, it might require further development in car calculations that are steering, brake and gas. Also, the state machine might require more work, so it would have a modular system which allows anyone to create flexible states, by creating visual node system rather that writing hard coded states.

The final product can be seen in Figure 7.

Figure 7: 3 AI and one human player

Figure 7 was taken inside the final production, and contains three easy AI players and one human player.

9 REFERENCES

Astalo Games 2018. Retrieved 27.3.2018 http://astalogames.com/

Bourg M, David, Seeman Glenn. AI for Game Developers. O'Reilly, 2014

Gamasutra 2009. The Pure advantage: Advanced Racing Game AI. Retrieved 5.4.2018. https://www.gamasutra.com/view/feature/132313/-the_pure_advantage_advanced_.php

Raganwald, 2018, Forde's Tenth Rule, or, "How I Learned to Stop Worrying and love the State Machine". Retrieved 11.3.2018. http://raganwald.com/2018/02/23/forde.html

Unity Technologies, 2014, Documentation, Unityscripting languages and you. Retrieved 22.1.2018 https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

Unity Technologies 2017 a. Navigation System in Unity. Retrieved 13.3.2017. https://docs.unity3d.com/Manual/nav-NavigationSystem.html

Unity Technologies 2017 b, UnityScript's long ride off into the sunset. Retrieved 25.1.2018 https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/

Unity Technologies 2018 a. class-TimeManager. Retrieved 8.4.2018. https://docs.unity3d.com/Manual/class-TimeManager.html

Unity Technologies 2018 b. Colliders Overview. Retrieved 8.4.2018. https://docs.unity3d.com/Manual/CollidersOverview.html

Unity Technologies 2018 c, Standard Assets. Retrieved 31.3.2018 https://goo.gl/-AfM2qW

Yannakakis, Georgios N., Togelius. Julian *Artificial Intelligence and Games*. Springer, 2018.