ELECTIVE GAME DEVELOPMENT REPORT

EL3 ELECTIVE GAME DEVELOPMENT (SEMESTER 2)

Umberto Falkenhagen – Student number 428796

CMV2A

Communication, Multimedia, and Game Design

11 June 2021

Teacher:

Armando Gerard

Contents

Game Development Elective – Snow Warriors

Report by Umberto Falkenhagen

GitHub-Repository: https://github.com/SnowWarriorsGameLabElective/SnowWarriors

GitHub-Repository (New Approach):
https://github.com/UmbertoFalkenhagen/SnowWarriorsUmbertoPriv

The task

The goal of the elective was to create a small game in which several self-programmed AIs would battle each other for the win. With 'Snow Warriors" we wanted to create a small battle arena in which the contestants would throw snowballs at each other and pick up collectables to stay alive, which should overall deliver a funny and interesting battle between them.

To effectively manage the work within our group we had several meetings throughout the weeks in which we discussed current tasks, split up the work and talked about issues. We used Trello to monitor these tasks and GitHub to manage the versions of our game when working on the code.

This version control allowed us to work on different branches when completing tasks and then merging them together once they work properly. Due to the current CoVid-19-situation we had to work and meet mostly online though since the half of this block we were also able to meet and work in person occasionally. Still, with using this version control system we were able to work quite independent from home.

The framework

The goal of the framework was to create a super-class that all AIs could use to execute different actions (e.g., moving somewhere, throwing snowballs, creating them, and dodging). By creating the mentioned super-class, we wanted to make sure that while all AIs would make their own decisions they would still be bound to the same limits, meaning for example that they would all run with the same speed (though they could choose between modes like running, walking, sneaking), take the same damage from the different projectiles, etc. Furthermore, the use of this super-class enabled us to create things like the UI-elements and a nice camera system without yet having the different Ais as we were able to just work with properties of the super-class type instead of the specific AI classes.

Originally, we wanted to also implement an interface, to make sure that all Ais would implement certain functions on their own (e.g., a function for deciding on their next action) but decided against that in the end as it did not make much sense under these circumstances to specify how the functions should be named.

To make it possible for the Ais to act depending on what is happening around them we also implemented a sensory system, which basically consists of constantly updated Lists with GameObjects of different types (e.g., a list for all collectables around them, one for all players they can currently see, etc.) that combined resemble the Players surroundings. The Ais could then make

their decisions based on the content of the lists, for example attacking a player that is in sight or collecting a health pickup that is close to them.

Polishing elements

As mentioned before, I implemented a sensory system (can be found as the EnviScanner script in the project) that is supposed to represent the different objects around each AI and stores them in easily accessible Lists. The Lists consist of different GameObjects (e.g., one list for obstacles, one for collectables, etc.) The sensory system is designed to automatically manage the content of each list, meaning that elements are added and removed based on their position to the AI that relates to each sensor unit. This means for example that enemies are added to the enemiesInHearingRange List if they are close enough to the measuring AI. If they are also in the FOV of the corresponding AI and not blocked by any other objects, they are also added to the enemiesInSight List. Once some of the criteria don't apply anymore the enemy in that example is removed from the related list.

Another feature that I added to the game was an animation management system. This allowed us to use different animations (e.g., throwing, creating snowballs, running) which the manager would adjust automatically to the players actions.

The individual AI

For my AI I created a priority system that consists of a List of possible actions and executes them based on their priorities. The priority weight of each action depends on the current events in the game and is updated whenever certain events happen. On the one hand, some of the actions are tied to certain events taking place, for example whenever an enemy AI enters the FOV it increases the weight of attacking enemies if there are snowballs that can be thrown at them, or otherwise increases the weight of retreating to a hiding spot in case there are now snowballs to throw. On the other side there are some actions that are tied to the stats of the character, for example picking up health collectables becomes more important if the HP of my AI is dropping whereas picking up stamina collectables becomes more important if the stamina is low.

Afterall, the goal of this system is to create a dynamically changing list that depicts the importance of the available actions and executes the one that is most important.

The pros

The advantages of the described systems are that it results in an AI that is very adaptive to its surroundings. In many instances the AI made decisions that are from a spectator's point of view mostly accurate, for example attacking when it has lot of snowballs but retreating if there are many enemies in sight but only few snowballs to throw. It also led me to an adaptive code that allowed for easily adding new actions and tying their weight to events and other gameplay elements.

The cons

The main problem that I ran into was the complexity of this whole AI model and its implementation into our existing game. As I did not plan on doing the AI like that from the beginning but only got to

that approach later into the project, I had to make a lot of compromises as I did not want to mess up the other Ais with changing things in the sensory system or the player super-class.

Another problem that made it complicated was, that to design a good working AI it was necessary to have in-depth understanding of event-based systems and a well-designed framework to work with. Though our framework was a good foundation and me being able to gather a lot of knowledge about that topic (and will continue working with such systems in the future) I had a hard time setting it up and had and still have to deal with a lot of bugs.

New Approach

Following some research that I did lately I wanted to try out another approach with an FSM (Finite State Machine). Unfortunately, this approach did not really fit into our existing framework or at least demanded some changes in our framework. To still try it out I created another GitHub-Repository and copied the existing project folder (the new script has the name UmbertoAINew). This enabled me to wildly try out this approach and change things in the player super-class (and accordingly also to change lines of code occasionally in the other AI scripts to adjust them to the performed changes). Though I did not have the time to get rid of the bugs that now occurred due to the changes that I made, I was at least able to create a working FSM.

I was also able to combine the existing approach of a priority weight list with the new FSM approach by creating a sorted list of the Interface type and adjusting the individual weight of each state to the events that are happening in the game. To make it efficient I would have to continue and tweak the state transition conditions as well as fixing the occurring bugs.

Summary

Reflecting on the work we have done it became obvious to me how complicated it is to create the concept and code for AIs. When we started creating our framework we went with a simple “inherit from super-class”-approach. Now after working on the project for quite some time, there are many things I would probably do different. However, based on the different knowledge that each group member had and still has, its always about making a compromise when developing the framework and AIs. In the future it should be a high priority to overhaul the existing super-class to reduce the bugs that are happening within its methods. Currently there are still plenty of bugs when it comes to managing animations as well as player states. Once the super-class bug fixing is done, I would resume tweaking the AI-elements or maybe also try new approaches.