Manoj’s speech on DDA and code reading

The Dynamic difficulty adjustment in flOw is a unique aspect of the game that alters creature interactions based on the player’s decisions. The calculated distance from the player and objects projects behaviors onto creatures depending on proximity and the player’s level of aggression. So, the creatures only attack the player if the player approaches the creature themselves. When the player attacks a creature, the creature is wounded. This changes the settings of creatures speed and target prioritization. If the player attacks the creatures, then the creatures attack the player at an increased speed. Therefore, the targets of the creature changes from food to the player. However, if the player chooses to wander around aimlessly through levels and undiscovered areas, they may wish to do so. In this case, the other creatures only target food or wander randomly throughout the whole plane. Another feature of DDA in flOw comes from the player cloned boss. This boss is an exact clone of the player, so it would look different depending on the number of segments the player has acquired through consuming food and other creatures. So, the more aggressive the player, the more health the cloned boss has. This is an impressive application of DDA in the game. The player has to fight or escape their own copy whose difficulty is determined by the player’s level of aggression. Nonetheless, the game does let you learn at your own comfortable pace. There’s a large open world to explore in flOw if players seem uncomfortable with the game mechanics or tackling obstacles such as other creatures or bosses. This feature lets players experience flow without being chased and creates a relaxing experience. Nonetheless, players can be as aggressive as they want to be by jumping through levels and devouring creatures at their quick pace. FlOw opens up a world of exploration in deep waters for both inexperienced and experienced players.The dynamic difficulty adjustment of flow challenges players but avoids the frustration that is present with games that have a set difficulty. Furthermore, if a player seems to be having trouble continuing, the difficulty is decreased, but if a player is doing well and progressing smoothly the difficulty is increased.  The dynamic difficulty adjustment of flOw caters for all types of players.

**Code Reading:**

Reading through the code for these segments of the program seemed to be quite a challenge. I’ve alleviated some of the problems by approaching the code in a more organized manner than just jumping straight in. After listing out all the methods in the classes I was focusing on, I chose six of the methods that looked like they were central to the program. Learning these few functions gave the big picture of the program and made the rest of the functions easier to place in the overall purpose. Working from smaller parts and onwards has also helped in understanding each of these separate functions as well. As I kept reading through this code I paid close attention to the function names that were being used. This gave a whole overview of what I will be looking at through a particular function. For example, Reading the graphics code with no specific and immediate visual example available was difficult to comprehend. I took a look at the function names for each of the function to understand what their specific purpose is. One function might be dealing with the blur of an image while another will be dealing with shading. The clear function names and comments really helped with understanding the code. The fact that the code was written in actionscript proved to be quite a challenge as well. Here, I looked for familiar logic and data structures I’ve used in other languages to help me navigate through the code. Navigating the code was made easier with the simple comments I saw at several places before and sometimes within functions. They seemed to be reminders more than comments as to what is happening throughout the program. Another strategy I used that helped with understanding the code was iterative reading. This process lets me go back to review previous parts of the code after learning new information. Knowing more information about the code and rereading other segments helped me understand the program thoroughly.  A crucially important part of the whole program is the creatures class. Aaron here will be talking about the creatures in the game.

DDA

The Dynamic difficulty adjustment in flOw is a unique aspect of the game that alters creature interactions based on the player’s decisions and ability. If the player seems to be having trouble and gets eaten by a creature they aren’t killed but sent back a level. The lower the level number the easier it is to survive. However, the player can go be as aggressive as they want by jumping through all these levels.

Another interesting aspect of flOW is that the calculated distance from the player and objects projects behaviors onto creatures depending on proximity. The other creatures are set to wander around in the oceans prioritizing food as targets. However, when the player attacks the creature, that’s when the creature’s status is set to wounded and its movement attributes change. Such attributes consist of its target prioritization and movement speed as found by the use of the panic intensity variable in the behavior class.

My main focus on this project was the graphics of the objects and the behavior of the creatures. My experience with reading through these classes wasn’t exactly ideal.

documentation

some comments were bits of code that were probably supposed to be removed.

The comments didn’t seem to help with reading the code. They felt more like reminders to the programmer themselves. One comment I’ve come across was a one-word comment that just says egg. These comments weren’t used effectively throughout the code.

general readability

However, the function names and variable names were very specific and clear. This allowed quick understanding of what was happening in functions. These names were given as an exact description of events or behaviors of the creatures. Like the variable coward or the function name startBoost in the variables class.

The indentation and placement of brackets of the logic was consistent throughout the code. This allowed for an easy trace of the logic without having to find where something started or ended.

Code reading reflection

One of my main strategies for reading this code revolved around the clear and descriptive variable names. After listing out all the methods in the classes I was focusing on, I chose several of the methods that looked like they were central to the particular behavior of the creatures and the player. I traced and understood how these variables were manipulated throughout the program. Learning these few functions gave the big picture of the program and made the rest of the functions easier to place in the overall purpose.

Working from smaller parts and onwards has also helped in understanding each of these separate functions as well. In this strategy I read the code and understood as much or little code in the functions. Understanding even little pieces provided for a stepping stone to understand other sections of code. It was an accumulative process that eventually lead me to understand the program piece by piece.

A strategy that didn’t work so well was iterative reading. Given the large mass and expanse of code in each class and function, reading and re reading only used up more time and proved to be very inefficient.

structure

language

Actionscript.

I’ve never read or written in Actionscript.

It was very object oriented and similar to java. Minor differences such as its

These features of the language didn’t hinder my understanding of the code too much after getting used to.

coding style

Manoj’s speech on DDA and code reading

The Dynamic difficulty adjustment in flOw is a unique aspect of the game that alters creature interactions based on the player’s decisions. The calculated distance from the player and objects projects behaviors onto creatures depending on proximity and the player’s level of aggression. So, the creatures only attack the player if the player approaches the creature themselves. When the player attacks a creature, the creature is wounded. This changes the settings of creatures speed and target prioritization. If the player attacks the creatures, then the creatures attack the player at an increased speed. Therefore, the targets of the creature changes from food to the player. However, if the player chooses to wander around aimlessly through levels and undiscovered areas, they may wish to do so. In this case, the other creatures only target food or wander randomly throughout the whole plane.

Another feature of DDA in flOw comes from the player cloned boss. This boss is an exact clone of the player, so it would look different depending on the number of segments the player has acquired through consuming food and other creatures. So, the more aggressive the player, the more health the cloned boss has. This is an impressive application of DDA in the game. The player has to fight or escape their own copy whose difficulty is determined by the player’s level of aggression.

Nonetheless, the game does let you learn at your own comfortable pace. There’s a large open world to explore in flOw if players seem uncomfortable with the game mechanics or tackling obstacles such as other creatures or bosses. This feature lets players experience flow without being chased and creates a relaxing experience. Nonetheless, players can be as aggressive as they want to be by jumping through levels and devouring creatures at their quick pace. FlOw opens up a world of exploration in deep waters for both inexperienced and experienced players.

The dynamic difficulty adjustment of flow challenges players but avoids the frustration that is present with games that have a set difficulty. Furthermore, if a player seems to be having trouble continuing, the difficulty is decreased, but if a player is doing well and progressing smoothly the difficulty is increased.  The dynamic difficulty adjustment of flOw caters for all types of players.

**Code Reading:**

Reading through the code for these segments of the program seemed to be quite a challenge. I’ve alleviated some of the problems by approaching the code in a more organized manner than just jumping straight in. After listing out all the methods in the classes I was focusing on, I chose six of the methods that looked like they were central to the program. Learning these few functions gave the big picture of the program and made the rest of the functions easier to place in the overall purpose. Working from smaller parts and onwards has also helped in understanding each of these separate functions as well.

As I kept reading through this code I paid close attention to the function names that were being used. This gave a whole overview of what I will be looking at through a particular function. For example, Reading the graphics code with no specific and immediate visual example available was difficult to comprehend. I took a look at the function names for each of the function to understand what their specific purpose is. One function might be dealing with the blur of an image while another will be dealing with shading. The clear function names and comments really helped with understanding the code.

The fact that the code was written in actionscript proved to be quite a challenge as well. Here, I looked for familiar logic and data structures I’ve used in other languages to help me navigate through the code. Navigating the code was made easier with the simple comments I saw at several places before and sometimes within functions. They seemed to be reminders more than comments as to what is happening throughout the program.

Another strategy I used that helped with understanding the code was iterative reading. This process lets me go back to review previous parts of the code after learning new information. Knowing more information about the code and rereading other segments helped me understand the program thoroughly.