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. The wounded parameter changes the settings of creatures to increase speed, turn speed, and targets. 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. 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. There’s a large open world to explore in flow if players seem uncomfortable with the game mechanics or tackling obstacle 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 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 DDA 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 classes 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 parameters of each of the functions as well. This gave a whole overview of what I will be looking at through a particular function. For example, Reading the graphics example 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 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 increased 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 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 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 classes 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 parameters of each of the functions as well. This gave a whole overview of what I will be looking at through a particular function. For example, Reading the graphics example 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 fucntions. They seemed to be reminders more than comments as to what is happening throughout the program. A crucially important part of the whole program is the creatures class. Aaron here will be explaining the many creatures and their defining attributes.