Sigi Lopez
05/11/18
Professor Kyrilov
TA Mina

Final OOP Project Report

Introduction

Our team name is Game of Drones, we decided to create a video game for our final project. We mostly made this decision because it seemed the most interesting to us from the multiple options we had. The game that we decided to recreate was pong. Since we knew that just creating a black and white pong would be too boring, we added some new features and made the graphics interface better. To create our game, we knew that it was going to involve a lot of the material we have learned throughout this class. Object oriented programing solved a lot of the problems that we faced thorough making this game. We decided to call our game Pongnite because it sounded funny and unique. The style of the game is supposed to be retro, vapor wave style which assimilates to when pong was mostly played. Another addition to creating just the basics of pong. We decided to add another feature which is power ups. These power ups changed the whole setup of the game. First of all, it made the game more fun and it also made the artificial intelligence less agile. Sometimes our Artificial intelligence seemed to be to good and since we needed to keep a balance we didn't wanted to make it too bad at the game. Our game includes all the basic needs for a classic video game which is start menu, score counter, pause button, save button, load button and a restart button.

Use of Object Oriented Programing

```
class Title {
    float x;
    float y;
    float w;
    float h;
    GLuint texture_map_id;

    int rows;
    int cols;

    int curr_row;
    int curr_col;

    bool complete;
    bool animating;

public:

    Title (const char*, int, int, float, float, float, float);

    bool done();

    void draw();

    void advance();

    void incY();

    void reset();

    void animate();

    void stop();
};
```

```
class Paddle {
public:
    float x;
    float y;
    float w;
    float h;
    GLuint texture_id;


    Paddle(const char*, float, float, float, float);

    void draw();

    bool contains(float, float);

    void moveUp(float rate=0.1);
    void moveDown(float rate=0.1);
    void moveLeft(float rate=0.1, bool free = false);
    void moveRight(float rate=0.1, bool free = false);
    float xCenter();
    void bot(float x);

    bool movingLeft;

    float xinc, yinc, translate, botSpeed;

};
```

One specific part where we needed to include object oriented programing in our game was when we started dealing with textures. The textures were very easy to implement when the game was very simple which meant that we weren't using anything that animates. Once we started getting more in depth through the project we decided to make different textures appear as objects in the code. This solved a lot of our problems when we were trying to implement

specific features to animating textures, but it was causing problems if we wanted to use the same type of object with non-animating textures. For example, some features we only needed for animating textures was like skipping animating frames, stopping and resetting animations. As you can see in the image above there's two different type of classes one of them is called a tittle and the other is called a paddle. It makes sense that we needed to make different object types because their functions are very different.

Another way we used object oriented programing to our advantage was to make our code more organized. It seemed to be very amazing to just write one line of code in the app.cpp file. Since the beginning of this project we knew that the structure on how our objects worked was really important. This is the main reason we created them in such a way that everything comes from the main object which is game. This specific type of object programing is called composition. The only file that the object game interacted with was the app.cpp file. Doing this made our programing more understandable and easier to modify instead of just cramming everything to the app.cpp file.

```
class Game{

public:

    Game();
    void draw();
    void drawScore(char score,char player, int length, float x , float y);
    void animate();
    void clickControl(float x, float y);
    void specialKeyPress(int x);
    void specialKeyUp(int x);
    void keyPress(unsigned char key);
    void keyUp(unsigned char key);
    void pause();
    void save();
    void load();
    void reset();
    ~Game();

    bool game_over, up, down, left, right, a, d, moving, startMenu, gamepause, started, usingPower;
    int playerMode, bounces, whoPower;
    float Power_Up_left_start_x, Power_Up_right_start_x;
    char playerOneScore, playerTwoScore;

    Title* title;
    Title* gameOver;
    Static* background;
    Button* onePlayerButton;
    Button* twoPlayerButton;
    Ball* ball;
    Paddle* Player_1;
    Paddle* Player_2;
    Static* Player_1_Win;
    Static* Player_2_Win;
    PowerUp* Power_Up;
```

In this code above, it is shown that we decided to create our objects in the heap which is also called the dynamic memory. There were many advantages of doing this, particularly the best one is that we can change data more freely. Storing information in the heap is more efficient and makes our game run smoother by implementing our memory usage carefully.

Every single time we used the instantiation new we knew that we needed to use delete to free the memory.

Time Plan and Division of Labor

At first our team seemed to be detached since we haven't had that much experience working together but as time went on we learned how to finish different tasks together. Since the game was very complex we split our team into the three team members that we had. Juan dealt with all the games physics and logic, while Alex dealt with the power ups. I mainly dealt with the object oriented organization of the code and the graphics of the game. The first problem we had was when we had to initially work on the programming. Since some of us couldn't work on specific features since we were waiting on another teammates work, sometimes the best thing we could do was to wait. This specially happened when we had to initially create the physics of the game. Me and Alex waited until Juan finished so we could begin our tasks. Although we never ended up having conflicts this seemed very frustrating at first. Some of the tasks couldn't have been done specifically from just one person so we were very open to accepting help from eachother.

Lessons Learned

As I found that working as a team brought many advantages to how I learned the more I enjoyed working on this project. I certainly learned how hard it is to collaborate on programing since it may not be so easily to just work on your specific task until somebody else is done with theirs. Another important skill that I learned from this project was to ask for feedback on how you are programing something from your teammates. This helped me so much when there were times I didn't know where my bugs in the code were. The most important skill that I learned was to create a whole object oriented structure from scratch. Ever since I have learned Object oriented programing I have used it a lot outside of this class. It seems like it has unlocked a lot of the basic experiences I needed to complete higher division classes. I know that this class will be paying off specially on my careers future.