PIRATE PINBALL

Alex Rinaldi
Omar Oseguera
Hassen Seid
Software Life Cycle Models

The main model used in the development of *Pirate Pinball* was the **Waterfall Model**. The Waterfall divides software engineering into a series of steps: requirements, design, implementation, verification, and maintenance. The waterfall model is sequential; this means that each step has to be fully completed before the next. The result is a complete product that matches the original vision, but as we found during development, it is difficult to conceive all of the requirements and features of a product in the beginning of the process. The Waterfall model was useful for features that were planned from the beginning of development, such as physics and animation.

For features that we added later in the project’s life cycle, we used a modified version of the **Agile Model**. The agile model uses the same basic steps as the Waterfall model, but it is iterative - this means the steps are repeated over and over for new individual features. This allows the project to be more adaptable to new ideas, along with allowing more frequent releases to consumers. Our version of the Agile Model was less structured; we did not use sprints or unit tests, for example. In future projects, we can try to incorporate these components.

**Teamwork**

An important part of developing software is working on a team. We tried to incorporate some ideas from the Agile Model to help our team to be productive. One idea is “**pair programming.**” where two or more programmers work side-by-side on the same task. Pair programming allowed one team member to come up with more general ideas, while the other team member decided how to implement the ideas. For example, this helped us come up with a solution for pausing the game.

Another important idea is the **SCRUM stand-up meeting**. While developing the game, our team had brief meetings every two days to discuss what each team member had worked on
individually, along with what each team member planned for the future. This helped us stay on the same page, and prevented team members’ tasks from overlapping.

Github
With github, it allowed our team to commit changes to our projects and viewing them from other computers. After pushing or uploading commits, our team members are able to view the changes made and the reasoning for the commits. Github enables fixing each other's mistakes much easier by being able to pull or download our project and making the necessary changes. The main purpose of github is allowing collaboration with team members to help build a better project which shows with our project. Other benefits github provided, is that it kept a history of all the commits made during the process of making our project. These feature acts like a backup just in case somethings goes wrong later.

The Waterfall Model Applied - Physics  by Alex Rinaldi

One of the features planned from the beginning of development was dynamic, realistic physics. Because of this, we were able to apply the waterfall model to its development. An overview of some of the steps used follows:

Requirements
Because much of the fun in a pinball game is watching the ball bounce and collide with the board, we knew that the physics for the game needed to be realistic and dynamic. After researching how similar pinball games were developed, we decided that the main requirements for the physics engine were accurate collision handling for rectangles and circles. Much of the components of the pinball board can be made from these two shapes.

Design
When designing and planning how the collision code would be structured, we knew it would be important for code to be easily reusable - most of the game objects share similar physics behavior and repeat many times throughout the board.

One way that we accomplished this was using modular code. The components for handling rectangle collision - detecting whether the ball is inside the rectangle, calculating the surface normals, and calculating the reflection vector - were placed in separate functions that could be called from anywhere. This made it easy to adapt the basic parts of rectangle collision for more complex objects, like cannons and flippers.
Implementation

We implemented rectangle physics by adapting classical physics concepts and vector equations. Rectangle collision, for example, is divided into three steps.
1) Detecting whether the ball is inside the rectangle
2) Detecting which side of the rectangle the ball is colliding with
3) Reflecting the ball’s velocity

Other game objects adapt the same physics concepts while adding others. For example, the flippers add the concept of torque, by increasing the ball’s velocity when it is farther from the center of the flipper.

Testing and Maintenance

Unfortunately, we were not able to adequately test the physics before finishing the project. There are some bugs in the physics, so this proves how important these phases of the life cycle development model are! One way to test the physics code is with Unit Testing, but since there are so many possible scenarios to test, a form of stress testing may be better. This may involve bombarding a rectangle with balls at many different random velocities.
The Waterfall Model Applied - Animation and Sound by Omar Oseguera

Other features planned from the beginning of development were the use of sprite animations and the use of multiple sounds. Planning these features from the start allowed for the application of waterfall model by setting a specific set of requirements.

Requirements

**Animation:**

Pirate Pinball was decided from the start, so the pirate theme was a driving force in decisions related to animation. We made it a requirement to create a pirate-ship world, and to animate aspects of a pirate ship such as: flags, cannons firing, a treasure chest, ropes, and a ship wheel.

**Sound:**

Our game was heavily influenced by the Windows Space Cadet pinball game. In Space Cadet, there is an array of sounds corresponding to objects on the board. We sought out to recreate that feature, and therefore made it a requirement to let multiple sounds play in our game.

Design

**Animation:**

Designing animation was more than designing code. Animations are sequences of images running frame by frame at a certain rate. In order to accomplish this we needed to find sprites, or make them ourselves out of images. For coding, we wanted to make animation functions generic, so that all animation-sprite sequences could be created via the same functions. In doing so, we would not have to continuously write the same OpenGL code!

**Sound:**

Sound design was driven by our requirement of having multiple sounds playing in the game. This meant designing code that allowed multiple buffers, sources, files, and a naming functionality. The naming functionality would allow all programmers to play whatever sound they wished by only one function call that takes a sound name. We also needed to design our sound such that only the appropriate sounds would loop, while others were one-shot sounds.

Implementation

**Animation:**

Animation implementation consisted of creating appropriate sprite animations, or building off of sprite sequences found online. During this implementation, we used a timer in our code to slow down our frame rates, and started/ended our frames with blank sprites, causing an appear/disappear effect. Additionally, animation required the use of the alpha channel, to which we assigned the color Black. By doing so, we were able to delete all backgrounds in our images.
Sound:
Given our design and requirements Sound, it felt best to use an Object-Oriented approach. Thus, a Sound Class was created. The default constructor loaded the appropriate sounds and names. Other key functions loaded buffers and sources by iteration. One of the most important aspects of the sound class was the optimized \texttt{playSound} function. Once an object is instantiated from the class, all the programmer needs to do is call the \texttt{playSound} function like so:
\begin{verbatim}
playSound((char*) “soundname\0”);
\end{verbatim}
This implementation exemplifies one of the benefits of object-oriented programming. The programmer calling the function only needs to know the name given to a sound, and does not have to worry about indexes in arrays.

Testing and Maintenance

Animation:
Unit Testing was used to solve an animation problem the team encountered.

What scenario were we testing? - We were testing the speed of a sprite animation loop.
What is the function \textit{supposed} to do? - The function is supposed to animate the sprite at a reasonable speed so that the animation is actually visible on screen.
What does it \textit{actually} do? - The animation was too fast!

Our tests involved using a timer to find the right difference to use when changing from sprite to sprite. We found our solution (1/20th of a sec), and the animation is at a reasonable speed.

Sound:
There was not much testing required for sound, as the design and implementation handled all appropriate situations. The only sound-related problem we ran into throughout our development of the game was finding the appropriate place to loop our soundtrack. In the end, giving the soundtrack WAV file a loop boolean of TRUE was the solution.

The Waterfall Model Applied - score and menu by Hassen Seid

Requirements

Score:
Every pinball game has some type of point system and a way to keep track of it. Certain objects on collision, will give you a certain amount of points and so the player is rewarded every time they play. The points the player receives is then incremented by a certain number to have the rising effect.

Menu:
Pirate Pinball has the same menu features as every other game. Pirate Pinball comes with a main menu, pause screen and a game over screen with a list of options to choose from.

Design

Score:
The plan was to design a point system that gave each object its own number value. After the ball collides with objects on the table, the points received are then totaled out. In order to do this we had to create a structure called Score that has three variables in it called points, rising points, and pinballs left. With a function called addScore, we are able to create a score object and add up the points together. With another function called risingScore, we were able to give the numbers that rising effect by incrementing the score which is displayed onto the screen.

Menu:
Every game has a menu and so we decided to follow the tradition and create a main menu, a pause menu and a game over menu. In order to make these menus, we used image editing software and created these images with a list of options. Then we used the keycheck function to make each option active and ready to be picked. For example, on the startup of pirate pinball the player has an option to start the game by pressing ‘1’ or exit the game ‘2’.

Implementation

Score:
When score is implemented:
1) Score counter starts at 0

```c
Scorekeeper.points = 0;
Scorekeeper.balls_left = 3;
```

2) On collision with objects for example:

```c
f.state = 1;
addScore(&Scorekeeper, 10);
```

3) Points added:

```c
void addScore(score *s, int add)
{
  //Adding score points after collision
  s->points = s->points + add;
}
```
4) Points are recorded and Displayed:

```c
risingScore(&Scorekeeper);
ggprint16(&re, 22, 0x00ff0000, "Balls Left: %i", Scorekeeper.balls_left);
//displaying the points in increments of 10
ggprint16(&re, 22, 0x00ff0000, "Score: %i", Scorekeeper.rising_points);
```

**Menu:**

When running the game, the first thing you see is a traditional main menu screen. Photoshop allowed our team to get the perfect image we needed. Throught a char array, the image then go to a for-loop where the picture gets converted to a ppm file.

For Ex)

```
for (int i = 0; i < NUM_IMAGES; i++) {
    strcpy(filename, ImageFile[i]);
    char *period = strchr(filename, '.');
    *period = '\0';
    sprintf(syscall_buffer, "convert ./images/%s ./images/%s.ppm", ImageFile[i], filename);
    system(syscall_buffer);
}
```

**Testing and Maintenance**

**Menu:**

**What scenario were we testing?** - We were testing the options on the menus to see if they work correctly.

**What is the function supposed to do?** - The options on the menus screen are supposed to take you where it said it would

**What does it actually do?** - The menu options were not working right, so we had to adjust the menu options with the keycheck and we had to make sure it was going to the right places
Score:

We did not have any test to scenario to do for the score because all we have to do was assign a value to an object and once the ball hits the object, so the points are being added together.