SHOOT SOME OTTERS PROPOSAL

Helen Oleynikova | Chujiao Ma | Jared Barrow

THE PROJECT

The general idea for the project is to make a shooting game that shoots otters in various ways. We intend to make it 2-dimensional shooting game, where you are given a gun to shoot as many otters as you can within a time limit.

The very minimum presentation would have at least 3 levels, each with a different graphical background, with otters moving around on it at various speeds. It would employ a point-and-shoot interface, much like whack-a-mole. The maximum presentation would involve more complexity and components to each level to make the game more interesting: such as different otters, different otter behaviors, or different guns in addition to changes in speed. The maximum presentation would also include features that make the game more complete such as high score table, instruction page, and aesthetically pleasing graphics.

DESIGN DECISIONS

One design decision that we faced was the choice between Tkinter and pygame. Both have their individual strengths. Tkinter is one of the most popular GUI interfaces for python and for good reason. It handles the layout of buttons and canvases well and is able to handle window resizing. Additionally, it handles drawings as objects, which means moving them around the canvas and so on is trivial. However, pygame is built for the type of program that we hope to design, it is excellent for making 2D games. It is lower level and less abstract. While we have to handle redrawing the canvas at each frame ourselves, we can easily create and edit our game objects with the Sprite class. Pygame also has methods to handle the collision detection. Since we chose to use Pygame, which only handles 2D graphics well, it lead to another of our design decision, which is to make Shoot Some Otters a 2D game instead of a 3D game.

We decided to forego making a 3D game mainly because of the added complexity, not because we chose Pygame. While technically anyone can draw a picture and pull up a photo for a background, modeling is a lot more complicated and time-consuming. 3D game with collision detection and such would not only be really difficult to implement, but would also be very slow and take a long time to run, which would be inconvenient for our game.

PREVIOUS DESIGN REFINEMENT

One of the problems we encountered was the need to display different objects during different stages in the program. Examples include a screen to show a menu of options like “Start New Game” and “Instructions”, a game over screen, and a screen to actually play the game. Trying to implement these different screens in the main loop of the program would be difficult and messy. Instead, we decided to create an abstraction called a *Screen* class: essentially, just a class that defines a few functions every screen should have. The Screen class has two methods: *update*, which will be called when the screen needs to refresh, and *handleevents*, which should process any events that may happen in a way appropriate to the screen.

All the specialized screens then inherit from the *Screen* class and override the *update* and *handleevents* functions to suit the needs of the screen. For example, in the game screen, when a click occurs, the *handleevents* function of the game screen gets called and shoots an otter. In the intro screen, when a click occurs, the *handleevents* function instead presses a button and goes to a different screen.

With this abstraction, the main loop of the program can simply process the events and send them to the *handleevents* function and call the *update* function to make the window display updated information.

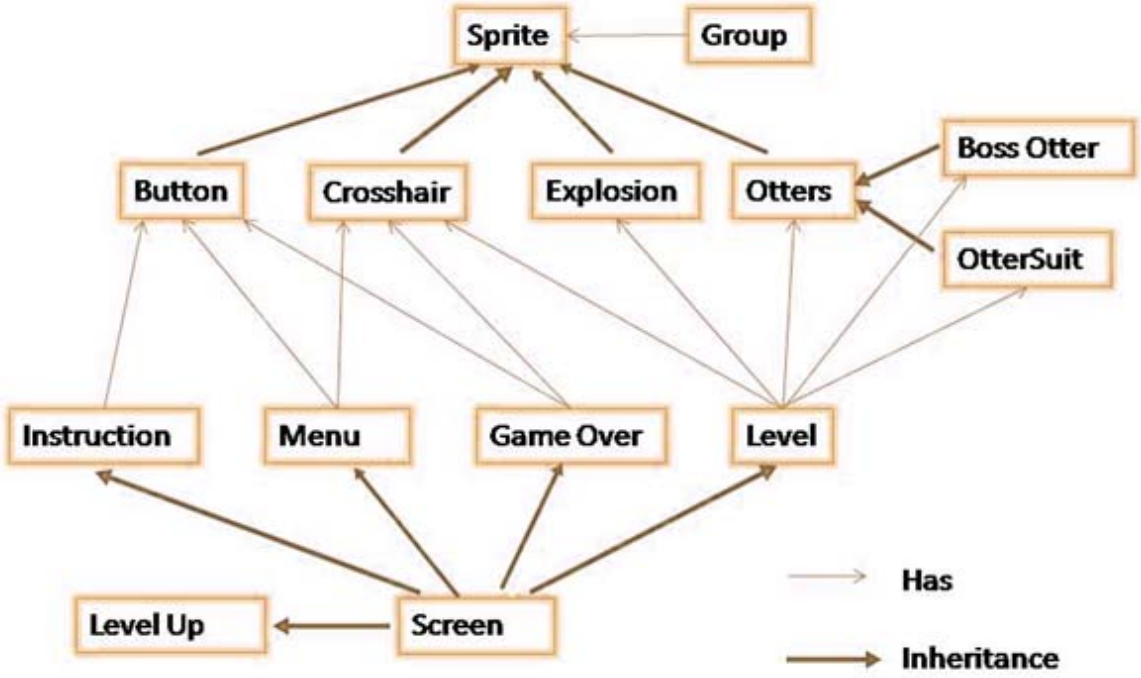
CURRENT DESIGN REFINEMENT

Major changes from the previous stage include the addition of the level up screen, the two new otters, refactoring how points are stored, and a graphical update. The original game just switched to the next level without warning when the time is up. The level up screen, which displays the points earned on the level and total points earned, was implemented to make the transition between levels less jarring.

The two new otters add new dimensions and strategies to game play. The minimum deliverable just had the otters moving at faster speed as the level increased. To make the game more exciting, we added boss otters at the second level, and fake otters at the third level. The boss otter appears at lower frequency than the normal otters, takes 3 hits to kill, but gives more points. The fake otters appear as often as the normal otter, and hitting one will cause you to lose points.

We refactored how the points are stored and split it into total points and level points. Before this change the player would be able to advance to the next level with firing a shot on the current level if they scored high enough on the previous level. With this change the level up function compares the score limit with points earned on the current level avoiding the previous scenario. We also changed the graphics (fonts, menus, toolbars and buttons) so the game has more aesthetically pleasing, and easily understandable interfaces as well as a more unified visual theme.

CURRENT UML CLASS DIAGRAM



DEVELOPMENT PLAN

Our process can be separated into three main phases. The first is to implement a basic working game screen, the second is to make different levels, and the third is to clean everything up and refine what we have.

In the first phase, we started by displaying moving otters on a background. We then went on to implement the game screen, where we would be able to click on the otters and record the points gained. Lastly, we set the conditions for the game. We implemented time limit and a top bar which displays time left as well as different points while playing the game. In this phase, we used hot seat coding: one person would be coding with the other two people looking over their shoulder, commenting and making suggestions. Then after a while we would rotate and do the same thing.

Phase two is where we implemented different screens and levels of the game. We all worked on different sections of the code. Jared made the start menu screen, Helen made the actual gaming screen, and Chujiao made the game over screen. We started the group work part of the phase after putting together the code for the screens. We wrote the code that calls different screens and then started debugging and writing new functions to make our implementation of the screens work. The way we wrote the code it was hard to implement these new classes without breaking the old functions of the code. The Screen objects such as Level Screen add an entirely different structure to our original code so it is simply a matter of reconciling these two warring factions. After we have accomplished these tasks, implementing new levels and screens will be a matter creating new screens with different attributes.

Now the game has all components for the minimum deliverable. We then set out to refine the game, make the interfaces more user- friendly and added new features that make the game more complete.

We did a major overhaul of the graphics. We changed the fonts used to fit with the more cartoony style of the game. Screen background and button background were created to have a unified theme for the game. A top bar was implemented to show the points and time limit clearly. A level up screen was implemented, different otters were implemented, and the behaviors of the otters were modified to create more variety in the different levels. We also cleaned up the code, commented the code, and in general prepared to freeze the code. We divided the jobs up, decided class time as the deadline for different jobs, and then brought everyone up to date during class, checked to see who did what, if the new version is uploaded, what we wanted to do next, and who is doing what.

FUTURE DIRECTIONS

We discussed many optional features while making the game. One of the options is, instead of a point-and-click interface, have the gun pivot in a limited arc depending on where the cursor is pointing, and have the player (and the gun) move from left to right with arrow keys. This would make for more interactive (and hopefully fun) game play.

Another addition would be: as you advance with levels, you could get new guns, and be able to switch between them in the game. The guns would have different properties, and hopefully get progressively better.

The game could also be expanded greatly in terms of levels: though the minimum deliverable is 3 levels, different mostly in just the background graphics, it could get more difficult progressively: faster otters, smaller otters, and otters with different properties. In the later levels, the otters could shoot back, which would further validate the moving left-and-right functionality described above. One more option is to unlock higher levels as you clear the current level.

The features that we are in the process of implementing is the high score table, where the user would be able to input his or her username and the game would record and show the top three scores. If we decide to take the project further after the high score table, then we can implement some of the options mentioned above.

BIGGEST PROBLEM

The biggest problem that we foresaw was feature creep. Though for the most part, we actually limited ourselves to reasonable goals between work periods, there was always the idea of “we should implement this,” or “we should implement that.” By keeping each other in check, we avoided having too major of a problem with trying to have too many features.

DESIGN ANALYSIS

We currently are at the midpoint between our minimum and maximum deliverable. We have 3 levels with different backgrounds and increasing difficulty. We also have more than one otter type: we have the normal otter, the boss otter and the fake otter which possess different point values and challenges. A different otter is introduced at each level. As stated before, the boss otter takes more

shots to kill, and the fake otter subtracts points when shot. These new otter types introduce a new aspect to the way the game is played: instead of shooting everything moving on the screen, the player has to be slightly more discerning. In addition to the 3 levels we have the level up screens, main menu screen, instruction screen, and game over screen. We worked on the graphics and made the game more aesthetically pleasing and more user-friendly. We are currently working on the high score table. We did not implement different guns in the game, and the game could be expanded with more levels. We feel that we had a realistic expectation of our goals and able to accomplish them in a timely fashion.

Design Reflection

Perhaps the best decision we made was to use the pygame module for our code. This shifted the difficulty from the mechanics of the game to the presentation of game and expansion of the game. We hammered out the gameplay fairly quickly, using the pygame sprite class as a basis for the otter and the crosshair classes, expanding them with attributes and functions that allowed them to interact with each other and exhibit proper behavior.

If we were to do it again we would probably split the code into several files. Instead of one 600-some line code file, we would have separated the classes into two or three different sections and put them in separate files, such as screen file, animated object file and text display file etc. This would have allowed us to work independently on the code with more ease, without interfering with the other persons work. However, it might make updating and version control slightly more confusing. We had discussed doing this but by the time that it came up, it would have be a larger time commitment, and felt we had more pressing issues to address.

Division of labor

As discussed in the development plan, at the beginning, we employed a hot seat method, rotating out of the seat with the others commenting along the way until we had our small proof of concept. Our proof of concept was a moving otter on screen that could be shot with a crosshair. After the basic is done, the hot seat method is no longer needed since the next step is to code several different components that can be done at the same time, such as the different screens. We would decide what needs to be done before the next meeting, then each of us volunteer to code different components. At the next meeting, we would go over what each of us did, make sure we all have the updated version of the code. If any one of us encountered difficulty coding his or her part, we all gather around trying to solve it during the meeting. We then merged the code and debugged as a group, decided what needs to be done next, then each of us volunteer to do a different task before the next meeting and repeat the process again. It worked fairly well. We all did approximately equal share of the work, and the meeting time brought us all up to date. Helen did slightly more coding than the rest, Jared did more proposals and cleaning up, and Chujiao did more graphics.

Bug Report

The Boss Otter was the biggest bug we had. It was the first type of otter inherited from the Otter class, and taught us an important lesson about inheritance.

Though we used the same functions for creating all types of otters, when we created a Boss Otter, at first, half of the time it wouldn't show up. Our first hunch was that it was spawning with part of it off-screen and killing itself, and after some quick testing, that turned out to be the problem. In order to counteract this, we made a function in the Otter class that, at initialization, checked if the Otter was fully on-screen. If not, the Otter teleported itself to a more suitable position.

After that was fixed, the Boss Otter always went in the same direction. We made sure that we hadn't simply broken something in the speed generation code, we tried making some normal Otters, and they still went in all directions.

Next, we thought it was because of the Boss Otter's huge size. The way the random speed generation works is that, if the otter is close to the edge, it picks the direction going away from that edge. We thought that since the Boss Otter was huge, it simply always triggered that check, and just went one way.

We did some testing with printing out the position of each corner of the image, and it turned out that even if it spawned on the right-most edge of the screen, the Boss Otter still went to the right, which made no sense. Frustrated and confused, we printed out each new Boss Otter's speed. When we ran the code, we got a stream of "(5, 0)".

The problem turned out to be that we relied on the Otter.__init__ function to initialize most of the parameters of the BossOtter class. When we called the Otter.__init__ function, however, we forgot to include the parameters passed into the BossOtter constructor, so it always set the speed parameter to the default argument. We fixed that line of code, and suddenly BossOtter worked perfectly.

Conclusions

We feel we learned a lot of things over the course of this project; we learned about object-oriented programming, figuring out how to use new modules, how to refactor code, and how to manage time and team management. On the whole, we feel we've accomplished our goals and shot some otters.