ChopShop Clicker Final Report

Participants

Piotr Jackowski - Project code, initial concept idea, research on current available game types done by all by myself.

Lorraine Hargrove - Play tested game throughout development, helped give ideas throughout development and thinking on the idea.

Abstract

I chose to create a phone based game. This game was meant to be a idling incremental game. It essentially is a game that takes minimal effort, so it can be played in short sessions. The player makes constant progress throughout playing, which can give it an almost addictive quality. Sometimes they are even fun. The game is usually really easy to play and learn. And the good ones contain a paradigm shift to change how the game is played.

Project narrative and Inspiration

All jokes aside this game is based off a lot of games. The most popular example I played is Cookie Clicker, which I played nonstop when it came out. Its simple concept with huge amounts of content and a constant growing curve to make the player take more and more work to climb a metaphorical ceiling which is usually connected to like a purchase that is completely out of your range. But eventually with time you will reach it.

A huge part of these games is that you can leave them off and when you come back the next day the game has progressed a little further. You now have enough money to buy the upgrade or research, so now you made that ceiling and completed your goal a new goal soon appears but now you make more than what you did before. This essentially continues along usually you reach a point you just cannot get past. At this point is when their is usually a shift in the game resetting all the progress but in return giving the player a bigger boost to get past that point.

There are a lot of these types of games on the market. Because of this I needed to differentiate my game to draw in different types of people and give a unique concept on the game. I realized I love my car and love other cars. Soon after I looked to see if there was a game like this on the market. I soon found that there is no idle clicker game made that is related to cars, while not being basic. So after watching car movies like fast in the furious, and especially *John Wick*, which intrigued me with the chop shop scene. Since that day, which was a while back, I made a mental note to make a game like this in the future. In which, you are in charge of a **ChopShop** where you not only steal cars but also Chop the car parts off to sell for money.

At first I planed to make it a phone game but since, I was doing the project by myself and also wanted to learn how to code in python. So I decided I would make the concept of the game in **Pygame** to not only be able to learn from your instructions on game design but also learn on a more fundamental level of game design. That way when I one day build and design this game in *Unity* or *Unreal Engine*. I would not only know how to build and design games but also have a fully planned out concept to go off and create a more visually appealing product. I am not saying that you cannot make a good looking game in **Pygame** but unity does make it easier by doing some heavy lifting, while giving shortcuts to a lot of code.

Initial player goals were to be able to learn the rules and how the game is played. Luckily idle games tend to be a lot more simple then most games so it is quick to learn. Immediately, players are looking to steal more and more cars to make more money. Soon they want it to go faster then what they can produce so they look for other options. The player will start testing certain things and realize that they can hire people to do work for them. However, the workers are slow at first so soon the player also wants to improve his workers while improving other things to keep things progressing. Eventually, the player will notice things will take longer so they will probably start saving to reach a certain upgrade like a multiplier or any other good worker/upgrade that will improve, in some what shape or form, the income of the player. This pattern of constantly reaching new levels even to the point of numbers so big it entices the player to keep on going.

Outline

Original Progress path

1) The initial Goal was to learn not only how to use **Pygame** but also **Python**.
    A) The first steps were to learn the syntax
    B) I need to learn how to make the game loops and where to place certain items and actions based on the loop to make changes to the game.
2) Learn how to create a button, which started with a very basic button
3) Create the buttons to know what game loops will be needed for the different types of screens
    1) Stealing - Meant to be a Map for stealing items.
    2) Inventory - Manage current part stocks for selling and possible future usage for upgrades.
    3) Research - For getting upgrades for the base and your income.
    4) Manager - Buy and upgrade workers.
    5) Events - Give extra incentive to keep on playing by giving random events.
4) I began working on the main game loop by creating and configuring the **Lifts**, which do most of the work.
5) Figure out a good data structure for my inventory and for my carlot.
6) Build a research page.
7) Manager and make workers upgradable.
8) Improve stealing and build a huge map.

9) Make a lot off different events and make them give different things.

10) Create a restart mechanism to add a paradigm shift, probably going to be a rebirth or something like that which gives the player better items or stats to let them progress even further in the game.

Formal Structures and Objects

There are quite a few formal elements contained in my project. The player themselves is able to do every task by himself and is able to interact with everything below. However, he can hire workers to do his job for him.

Car - Made up of a dictionary of parts which the workers, or the player can remove. Lift - Can hold both a worker and a car, in which the worker can work on the car to take parts off. Worker - Three types each pertaining to either stealing, selling, or working on cars. Workers stats can be upgraded through money to improve performance. Carlot - uses a simple FIFO queue to hold and limit the amount of cars a player can store, this of course can be upgrade in the future through research. Inventory - Uses a dictionary to store items and are sorted by both level and name, these parts can be sold for money. Research - Enables the user to improve essentially every part of the game to improve their income. These contain tech upgrades which are the most expensive components to research, these end up being the main player goals as this enables them to get to new places, better cars, and more money. Manager - Physically stores its workers and lets you interact with them. Stealing - Has a map in which their are locations to send your workers to steal cars or fail Events - These are timed to possibly go off every minute. However, they contain a chance number that way a player wont always get an event so they do not get boring quickly.

These events would be things like fast and furious to add some comedic relief and pop culture references that car fans would enjoy.

I would hope to add one they an actually store component of gang wars, this can be done through stealing by making a bigger map and adding territories to take over by defeating gang members. This would add a whole new level to the game by actually creating another thing that will be fighting against the player, like the huge price jumps.

Design

The game is built using Python and Pygame. Additionally I used Cx_freeze to build an .exe executable so I can share with people for convenience and easier testing. Specifically, so I can send it to my girlfriend for play testing. Shelve was also used to help save game components and to load them back in.

When you start the game, the first thing that happens is the initialization of every object and structure needed for the game. Afterwards the opening screen shows up where you can start the game, quit, or load the previously saved game. If you press start or load it will either load in the previously saved settings and items and launch the main loop or just launch the main loop. The first part of the game loop is to look and see if you loaded the game or not. The loop starts with checking the upgrades and other things like carlot to see if their values had changed. Since upgrades are saved this essentially repopulates most of that data needed. The game then goes through checking for events like mouse movement and clicking to see if these clicks or movements collide with my buttons created. This collision will simulate looking at a button, and the clicking event does the same, except also

calling a callback to a function which is predetermined when everything is initialized. This is done through the dictionary the button is made out of. Afterwards, there is callings to check if a worker finished on removing a part, or stealing is done, research complete, and if a seller sold something. This portion of checks is actually in every loop that way no mater what screen you are on your workers will keep on working. Lastly, there is printing statements for the buttons, carlot, lifts, and everything else.

The stealing loop does similar event checks on its buttons but is different in that the callbacks contain other functions imbedded. There is also different printing for obvious reasons and contains different stealing checks. Otherwise it is very similar to the main game loop. This is similarly true for the manager and research since they have similar layouts except the printing and checks are different for their respective job. Like research has to print out progress bars at different moments and needs to create buttons in different ways to enable researching. The manager prints and checks for worker purchases differently that way only the workers that are actually there are able to be upgraded. So if they are out stealing they cant be upgraded. That way things cant be broken through duplication and different stats.

If the saving button is pressed, the system first opens or creates a save file and the goes through every variable writing it to the file. Furthermore for a few of the variables it needs to convert them or get certain things out of them to complete the saving. Finally, it closes the file when done. Similarly, when loading it reads the variables saved and starts copying them into the game. With a few changes to make sure data is loaded correctly. For

example, images need to be reset and the game will check how much time has passed since you last played, which it uses to simulate gameplay to give cars and money to you.

Testing

One thing I regret not including is unit testing to make sure the base functions and stuff in my game is working properly. This could have saved me some headaches along the way by actually making sure each function does what is is supposed to. The only thing this wouldn't be able to test is actually the game loops, because they don't output anything and there would be no easy way for the tests to see if it was working properly.

The main source of testing was through play testing, which was done by both my girlfriend and I. It was crucial for me because it let me see not only if things were printing correctly but to find bugs. Some of which were incredibly hard to find because would happen at random moments and couldn't be recreated. When Lorraine play tested the game she would try everything and would report not only bug fixes with a print out of the line numbers of where things broke, but also gave ideas and improvements that can be done to make the game more enjoyable and fair. She gave me the idea of adding a timer to research, which made research more interesting. This research idea will let me in the future implement researchers which can do research for the player, but at a slower pace. This inturn will add even more content. Without play testing my game would be nowhere as close to what it currently is.

Limitations

Probably the biggest limitations on my project are purely because of me. I need to get better at art. Hopefully, by learning some skills in unity will help me to get better at making art, especially making animations like movement. Plus, overtime I will get better at *GIMP*. Another limit, is that I was doing the project by myself, while being simultaneously in six other classes. Even though this took a huge toll on me I still enjoyed every moment of the game design progress. Especially, once my game started coming together. A big limitation is my knowledge with pygame, because I struggled to save objects that contained pygame components. I did however find ways to go around this while still being effective. By disabling the images from being saved and simply reloading them with their correct color on loading.

I also struggled quite a bit with picking a good data structure for both the inventory and queue. The inventory was easier since, I had just learned about dictionaries in python and I almost immediately realized that they would be perfect for store car parts since they are dictionaries are easy to modify and don't need a lot of time to build and modify. The carlot was a lot more difficult, because I needed a queue, but queues wouldn't work because they don't function well in moments like asynchronous calls. Furthermore, they can caused a lot of problems with saving. I had to go through a lot of different data structures to only realize that queues are the best for what I need. However, I solved the problem by encapsulating the queue to make sure I was not removing when things were empty or adding when it was full. Furthermore, when I needed to iterate through the queue, which you normally can't do, I just simply copied the elements to a list through

popping. And after doing the work needed to by done I added the elements back. I still believe that this can be improved more, since the time order for coping and refilling the queue is `O(n^2)`, which can obviously improved to `O(n)` by using a data structure that isn't so finicky. I think a better option would have been to use a encapsulated linked list. I did try a linked list originally, but I think I just implemented it incorrectly, and that is why things weren't working.

A key missing feature is the lack of a completed set of content. This lack of content not only makes balancing even harder, since I need to almost predict on how the game will grow to balance the already existing items. However, I realize this is essentially how games are made, so as I progress further I will need to constantly need to think about balancing the game further. One thing that I am currently unable to do is a rebirth system, to create another paradigm shift. The reason I don't have one is because I don't know how to connect it into the game in a meaningful way. Futhermore, it is very hard to balance as is, by adding even more variables and things I won't be able to balance it at all. I think it is better to start small and grow outwards.

Lastly, I am missing a story for my game. While this wont effect the main mechanics, it will effect the stealing and rebirth mechanics. I plan to have the main character be someone who just got out of prison and wants to start his own gang, since his old gang betrayed him, which is why he is in jail. This story can add more because it will let me do gang fighting and sometimes your workers will be attacked and killed. This will add another dynamic to the game because certain areas will be high risk, high reward. So your

thieves can get very good cars and more exp but can be hurt or killed by rival gangs, which you need to defeat to take their territory.

Conclusion

Overall, I enjoyed making the more than I hated it for not working. The not working moments were surprisingly not as bad as some other classes "Distributed Systems" emulator problems. I not only learned a lot in **Python**, which I now feel comfortable using for whatever project I need. Also I understand why so many people like **Python**. It has very simple syntax, which dramatically improves readability. However, this doesn't necessarily mean my syntax will be easy to read (Sorry!). Furthermore, I now also know how to use pygame, which is a fun and simple game engine that can do so much. Plus I learned some deep mechanics of how games function, hopefully I can apply this to unity to let me create better games in the future. I understand how the code underneath the actual objects and game works, so I think I will be easily able to modify objects in unity, which I heard is the hardest part of unity. Its hard because the way unity interacts with C# is unique in someway shape or form.