Touch Crawler

Submitted by

Mason Chamberlain Turner

Computer Science

To

The Honors College

Oakland University

In partial fulfillment of the

requirement to graduate from

The Honors College

Mentor: Nilesh Patel, Professor of Computer Science

Department of Computer Science

Oakland University

November 19, 2020

Abstract

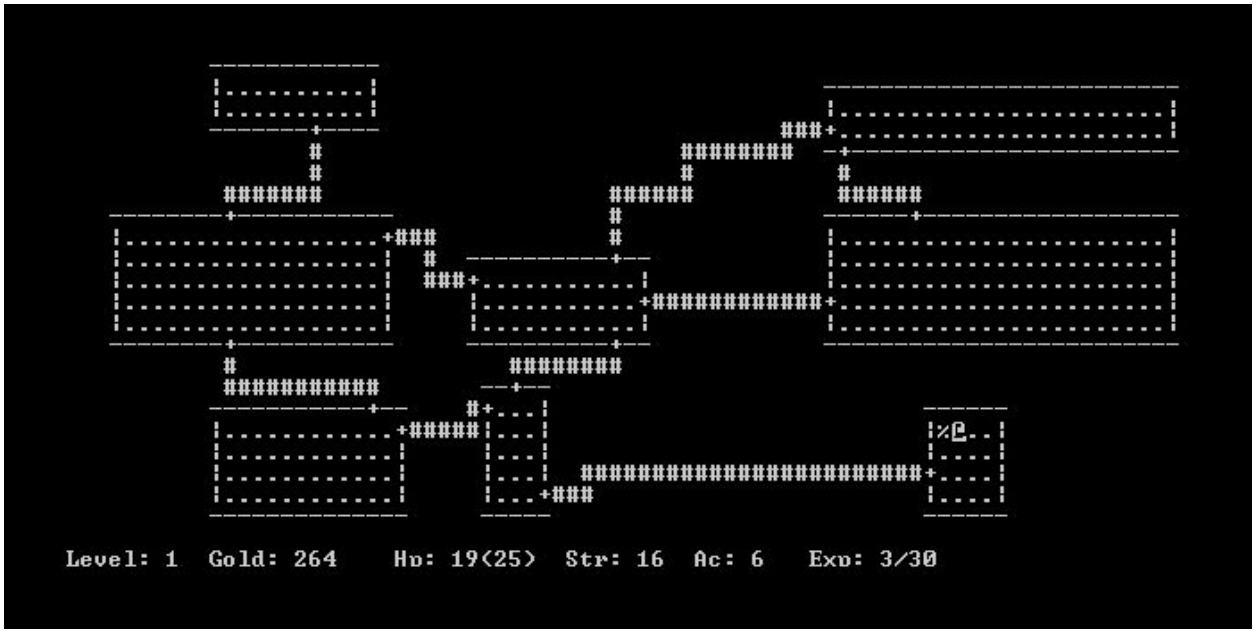
The aim of this project is to explore why the video game subgenre of roguelike dungeon crawlers differ in their control schemes between PC and mobile based games. This project will develop a roguelike game called Touch Crawler for the PC and mobile devices, in which it will offer a unique combination of genre and control scheme for the PC. Unlike most PC roguelike games, Touch Crawler will be mostly point and click as if it were a mobile game, while also being available for PC. Like most roguelikes, it will offer an experience of a scrolling shooter game where you fight your way through rooms, bosses and entire areas where the goal is to go as far as possible while winning the most points. Research such as this will be able to identify a possible new and innovative game design feature that may be used in roguelike Dungeon Crawlers in the future and answer the question as to why PC-Based roguelike Video Games do not use the same touch-based control schemes as their mobile counterparts.

Current Research

Rougelike dungeon crawlers (roguelikes) is a subgenre of role playing video games (RPGs) that was popularized in 1980 with the release of Rogue, developed by Michael Toy, Glenn Wichman and Ken Arnold (Parker 124). Rogue established many of the precedents of roguelike game design that are seen in the genre to this day. The player character in Rogue is an @ symbol, placed in a dungeon composed of text, where the map is generated procedurally, meaning that every time the game is played or reset, the map is completely random and different from the last. The player's objective in this simple turn-based game is to travel through each randomly generated level, defeat enemies (other text symbols), and go as far as possible (Garzia 1). Upon death, instead of being sent back to the beginning of a level or a checkpoint, the player is sent back to the beginning of the game. This feature is known as permanent death, or permadeath. These design elements can be found in many of the roguelikes that followed in Rogue's footsteps. Typically, roguelikes will use the standard WASD or arrow keys for control of the player character while using touch on mobile devices. Although there are some examples of roguelikes with mouse support such as Tales of Maj'Eyal (2012), these are much rarer than games which use standard PC control schemes.

Roguelikes increased in popularity until reaching their peak in the late 1980s and early 1990s. Researcher Maria Garda suggests that "it was the revolution in 3D computer graphics that took place in the early-to-mid 1990s that relegated roguelikes to their niche status." (Parker 124) Some roguelikes have been released for consoles such as the Playstation, Xbox and Nintendo consoles, but mainly they have been released on PC and mobile platforms. Another example of a popular roguelike is NetHack is one of the oldest video games to still be receiving updates to this day, despite being released in 1987 (Ho et al. 1). At the time of this paper, NetHack was last

updated in March of 2020, and that is unlikely to be its last update. Although they have waned in popularity since their peak, roguelikes have continued to be released and updated for decades. They remain a respected subgenre in the video game industry. Below, you can see a screenshot of the game Rogue in action.



Each of the core design elements serve a key purpose in roguelikes; The typical setting for roguelikes is in a dungeon, cave, underground labyrinth or otherwise closed off area. The settings lend themselves well to a sense of claustrophobia and adventure. The player character is often trapped in the middle of a complex web of rooms with no option to escape and few options to flee from enemies or defend themselves. When this happens, the player's stress level increases and they are incentivized to look for more health, power ups, or upgrades. Thus the incentive structure leads them to keep exploring and fighting more enemies. Procedurally generated maps

are meant to prevent boredom with the game, and do so by making the map different each time the player starts a new game or dies and is sent back to the beginning. If the player were to go through the same first level each time they died, they would easily grow bored or figure out strategies to easily make it through each level. Permadeath serves to add to the feeling of anxiety for the player. The goal of a roguelike is usually to go as far as you possibly can into the game. If you are in the middle of a dungeon and your character has low health, there is a constant mental threat of having to be sent back to the beginning of the game (Parker 124). When combined, all of these features create an incentive to replay the roguelikes. It is inherently satisfying to want to go as far as possible and beat your old scores when you cannot rely on knowledge of the map, saving the game or checkpoints.

According to software developer Jared Halpern, who specializes in Unity, “Game engines are software development tools designed to reduce the cost, complexity, and time-to-market required in the development of video games.” (Halpern 1) Game engines allow developers, whether by themselves or in large or small teams, to focus entirely on coding the video game. Many functionalities of video games such as graphics rendering, a physics engine for collision detection, an audio engine for sound and music and many more core functions (Halpern 2-3). These functions are provided by the game engines themselves so that developers don’t have to individually write code for each of them. This provides a tremendous advantage over coding a game entirely by yourself or in a team, because much of the work of creating the game was already done by the development team of the game engine (Halpern 1). Game developers thus need a game engine to develop their games in so that they may work as efficiently as possible. Unity is one such game engine. It offers development capabilities for both mobile and PC-based

video games simultaneously with the ability to scale projects for both PC and mobile devices (Takoordyal 110.)

GitHub is a repository hosting service meant to host software and make it publicly available to the world. Users can upload and download code, share it to the world, allow others in groups to help update it, and then upload an updated version of the software. This is very useful for large projects such as video game development (Bouquin 1). In other words, Github provides Version Control - "the management of multiple versions of a project" (Tsitoara 3.)

Version Control Systems save all versions and revisions of a project so that work isn't lost permanently. The benefits of this include the potential to have different creative visions be brought to light on a project, and the possibility of combining the best ideas from multiple different programmers and developers into a single project. Another benefit is that if a developer loses their work during a project, they can easily restore it to a recent state. If their computer shuts down while working, or if files get corrupted or accidentally deleted, then the developer may simply download their project to the most recent version again. Some work may be lost, but not all of their work. This makes it so developers do not have to start their projects over from the beginning whenever they accidentally lose their work, which saves tremendous amounts of time and stress.

Some of Github's most common functions include "fork", "fetch", "merge" and "pull" (Bouquin 1.) Each of these has a specific use in Github's main purpose. When a developer uses the "fork" function, they are using it to clone a repository so that the developer may work on a project by themself and make changes that they see fit, differently from their other team

members. Forking allows a developer to pull changes from another developer's repository as they personally see fit and give their own creative spin on a particular idea (Tsitoara 9).

Using "fetch" is the act of retrieving work done by other people, by copying a remote branch onto a temporary branch of a project. This allows the developer to edit the code on their own time with their own ideas. The "merge" command merges a temporary branch onto the main branch with the developer's own personal work done on the code. These two functions are the main method of downloading, editing and then working on code in Github (Tsitoara 210).

"Pull" requests combine "fetch" and "merge" into one. Using "pull" is requesting that their personal changes to the project be "pulled" and merged into the main project. A pull request does not necessarily need to be used on the original main project of a revision. A developer may dislike the direction that a project is going in, create their own fork, make their own revisions to a project, and then other users may create pull requests for the forked project.

Methodology

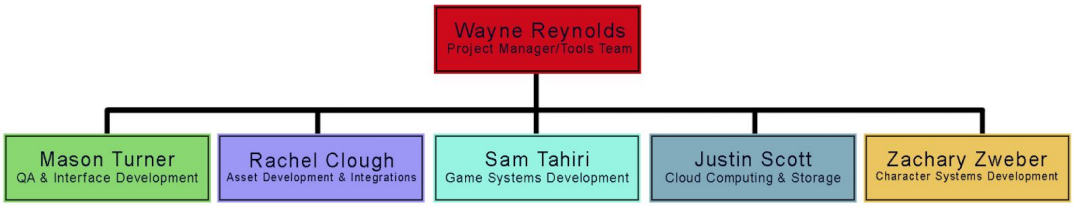
When Touch Crawler began development, it was decided that it would be developed in Unity. Unity was chosen for several reasons. Touch Crawler, like most roguelikes, was decided to be an overhead 2-Dimensional video game, and Unity is capable of developing 2D video games in a very easy manner. Unity also has an ability to develop games for both mobile devices and personal computers while using a single programming language, in this case C#. In order to edit the code, the team needed an environment that facilitated the usage of C#. Microsoft Visual Studio is an integrated development environment (IDE) that uses C#. It is used to write and debug scripts and files in C#. These files can then be saved and individually transferred to Unity.

Video games that keep track of user information and statistics, such as Touch Crawler, need to have a secure database. Google Cloud Platform was thus chosen because it allows for the hosting cloud software and information storage with security provided by Google. Specifically, AppEngine was chosen to host the application programming interface (API) for the game client. The database was stored in Firebase, and the authentication of accounts was handled by Firestore.

Every development team needs to be able to save their work, restore previous versions and have different team members work on different aspects of the project. Touch Crawler is no exception to that. Therefore the team chose to put Touch Crawler on Github so that the project could be managed and developed more efficiently. The inclusion of Github allowed the team members to work on their own project responsibilities at their own pace and on their own time. This allowed them to focus on responsibilities in their own lives such as coursework for classes other than the senior design project, for which Touch Crawler was developed.

Communication between team members is of the utmost importance in any project, and especially important for video game development. Without communication, each team member would have to try to figure out each individual development decision by themselves. Discord is a popular application/website that serves as a very easy way to communicate, collaborate and manage the organization of the project. Discord allows for both text and audio meetings, as well as the usage of bots to perform functions automatically. Touch Crawler's development team took advantage of this feature by writing a bot that tracked each commit to Github, so that it did not have to be manually logged.

The team that developed Touch Crawler was very small and thus had a simple organizational structure.



Each of these roles was not exclusive to each developer, as there was much overlap between each main aspect of development. Each developer worked on each main area before the final submission of the project.

Quality Assurance (QA) & Interface Development focused on testing the software while in development so that each change to the software worked as intended. Each function, line of code and edit was tested to see if the main game responded correctly to the changes. This was the main way that debugging was performed. Whenever a function was added, the team would start the game and see if the function was added and working as intended. The team also used this technique to decide whether the User Interface would be aesthetically appealing to users, or would appear at all in the game when booted.

Asset development & integrations was responsible for creating and importing art and sound effects for everything in the game. This includes character design, room design, textures, background objects, room layouts, firing sounds, and many other aspects. Art for the assets were created in GIMP (GNU Image Manipulator Program) and Photoshop. Both of these applications allow for the editing of images down to the level of singular pixels. Photoshop and GIMP were particularly important to the project because they allow easy editing of images down to the level

of individual pixels. Touch Crawler's art style was meant to resemble that of many retro video games from the 1980s to go along with its simplistic roguelike gameplay. Some examples of pixel art created in GIMP and Photoshop for Touch Crawler include barrels, tables, torches, rocks, boulders, pails with magic potions, boxes, enemies, powerups, weapons and the player character itself. Images created in GIMP and Photoshop were then added directly to the file folders of the main Touch Crawler project in the Unity folders before being uploaded to Github and then added to the main branch.

The standard workflow for asset development was a three-step process. First was identification of an asset that the game needed to be implemented. Such as a table or a rock for background art. This would involve team discussions to go over planned and in-progress features. Secondly, once a list of desired assets was compiled, it was given to the art development team and initial pixel drawings were created. Third, the art team would use GIMP and Photoshop to create more detailed artwork to be used in the final iteration of the asset and imported into the final project.

Game systems development concerned the development and implementation of different systems that affected the state of the game and the player's interaction with the world contained in the game. This included the rumbling of the screen whenever the player fires their weapon onto an enemy, thus giving the illusion of a real weapon being fired. This also includes world interaction mechanics such as moving from room to room via doors and the boundaries of a room.

The event system of Touch Crawler was designed as in a decoupling pattern and weapons system was designed as a monadic system. A decoupling pattern is when two objects in a game

interact with each other without “knowing” about each other. This allows developers to reduce their workload because if they want to refactor one system, there is less need to refactor the others. This provides a way for objects to communicate with other objects without needing a reference. In more technical terms, a sender such as an entity that wants to shoot at another entity, doesn’t need a reference to the receiver, the object being shot at. The sender can call methods on multiple receivers, such as the player character shooting at multiple enemies. There is information hiding between the sender and receiver such as the damage of the projectiles being shot or the health of the object being shot.

A monad is a pattern used to reduce the amount of repeating code. This allows for easy creation of different game assets that function similarly to each other. This was a very convenient set up for the weapons system because each weapon functions nearly the same as each other. Each weapon involved pointing and clicking on enemies to shoot at them, with the only major differences being the projectiles and damage dealt by each weapon.

Cloud computing & storage focused on server side development. This included setting up a server and building an API to interface with the game client and other third party applications. This was responsible for handling information about scores, profiles and other information that the team chose to share either publicly or privately. This information was held in a secure Google Firestore, which allowed for the team to have access to player information while allowing unique user identification to be handled entirely by Google. This had the benefit of both enhanced security and lowering the amount of work to be done by the team. Keeping track of the ID’s, passwords and statistics of players allowed for tracking of high scores so that players would have an incentive to beat their previous high scores and keep playing whenever they die and are sent back to the beginning of the game.

Character system development was responsible for the design and implementation of systems that the player directly interacts with. These things include the player character's movement, the enemy entities' movement, statistics such as health and weapon power, shooting weapons, and artificial intelligence. Each of these systems is integral to the gameplay and combat of touch crawler. Without them, the game would not work and would present no challenge to the players.

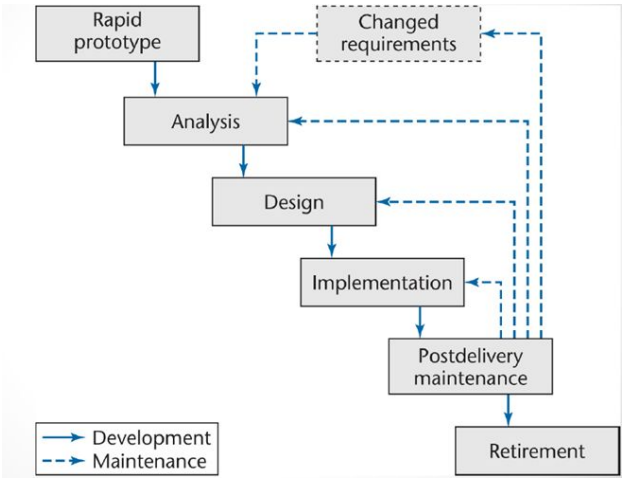
Project management & tools focused on taking the work of each of the other main systems and blending them into a cohesive and functional video game. Responsibilities also included creating tools in the Unity3D environment so that the amount of coding that needed to be done was greatly reduced. Assets, sounds and rooms could be added to the main project without having to code each one, making it so that images could simply be imported into the game, moved around and given statistics without having to individually code each object and its stats.

The first major goal of Touch Crawler's development was to set up the tools necessary to minimize the amount of work necessary to develop the game. This included setting up custom object editors in Unity and implementing design patterns to modify characteristics of objects on the fly. That way, everything else would be relatively easy.

The second major goal of Touch Crawler's development was to create a semi-functional demo with some content so that it could be playtested for bugs and to see whether the gameplay was appealing. The development tools established in the preliminary stages had some bugs, so they were ironed out after the playable demo was created.

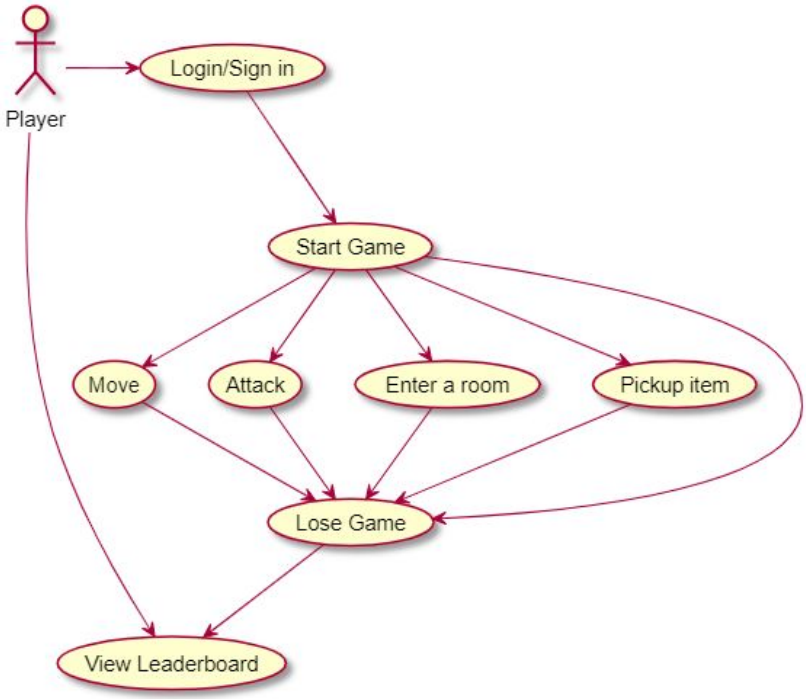
The third and final major goal of Touch Crawler's development was to add more content to the game, fix any bugs, add more items and lore, and make any last-minute additions and changes that were deemed necessary or desirable. Features such as ironing out the feel and sensitivity of the control scheme were added here.

Development of Touch Crawler was divided into seven "sprints", two-week long periods of time where the development team would set a goal for the development and complete that goal by the end of the sprint. The image above is an example of the model used for development - The rapid prototyping model. Given that most video games have much longer development cycles than a single college semester, teams need for prototypes to be created as soon as possible so that they may be playtested and receive feedback and scrutiny. Even on a smaller scale, this development model is necessary.



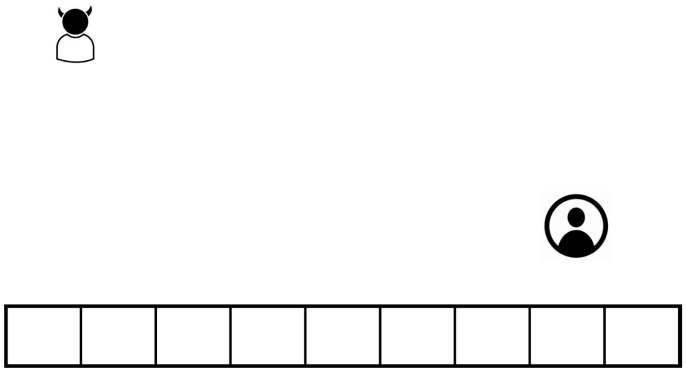
Sprint 0 of Touch Crawler's development came before the main sprints 1 through 6 and focused on preliminary work to set up a project vision and plan for the game's development. First, the team defined what a roguelike dungeon crawler is and their features. This included procedural generation of maps and permanent death. When the development team decided that

they wanted to create a roguelike dungeon crawler, they decided that the main objectives of the project, broadly, would be to move around, fight enemies, clear rooms and survive until the player character dies and then be sent back to the beginning. Whenever the player found a boss, they would fight against it and if victorious, they would advance to a different level in the dungeon. Subsequent levels would be procedurally generated just as the first one had been. Not only did the team decide to take inspiration from Rogue, but decided to also take inspiration from roguelikes with mouse support such as Tales of Maj'Eyal, Brogue and Cogmind. The team collectively decided, however, that Touch Crawler would only support mouse control.

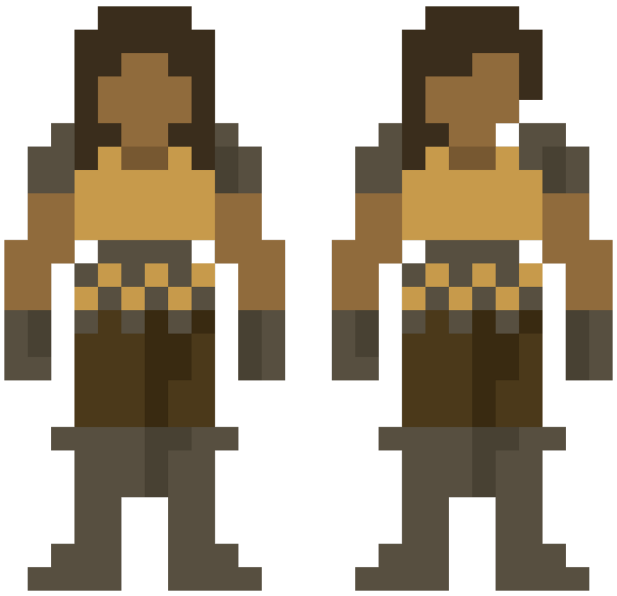


This simple use-case diagram shows a more specific set of use cases for the player and player character. The player can login or sign in to the game, begin the game, and then perform several actions within the game such as moving around, attacking enemies, entering different rooms and picking up items until they lose the game, defined as the player character dying. After that, the player views their score. The leaderboard feature was only available for mobile devices,

while scores on the PC version of the game are for the local device only. Below, you can see a preliminary design of the game that was created in order to give the developers an idea of what the final product would look like.

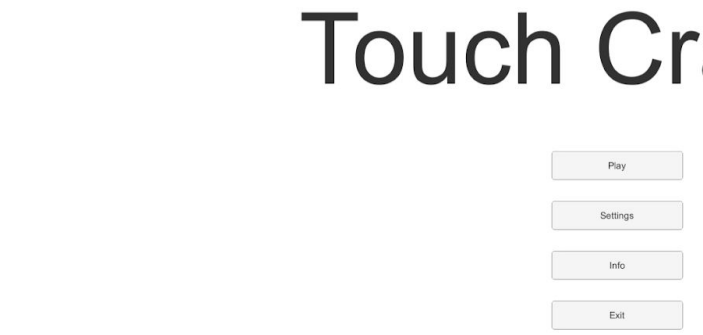


During Sprint 0, the team decided on an art style. The art style that was decided on was pixel art. Not only is pixel art relatively easy to create and edit, it also gives video games a feeling of being a retro game from the 1980s, when Rogue was released. Pixel art is a good fit for character customization. You can see an example of pixel art applied to this concept in the image below.



Sprint 1 was mainly focused on the establishment of Touch Crawler's login system and database. The development team decided to use the aforementioned Firebase and Firestore to host the databases and provide account authentication.

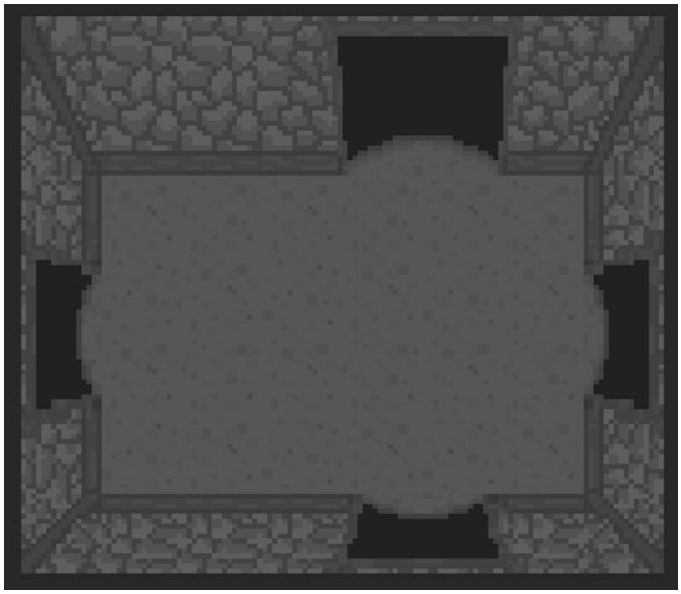
The user interface for logging in went through several iterations. The first one was a simple interface with a large text of Touch Crawler with buttons for "Play", "Settings", "Info" and "Exit." This gave way to a much more detailed and artistic title screen that more resembled the art style that was established.



During Sprint 2, the team focused mainly on the input system, event system and weapon system. This sprint is when movement was coded as part of the event system. A simple click will make the player character move to the point on the map that is specified, as it will be detected by the event system. The monadic weapons system was developed here, allowing the developers to quickly add new weapons into the game and focus on changing their attributes rather than coding each one individually.

The fact that everything in Touch Crawler is controlled by touch or clicks made it very easy to create the movement system for both the PC and mobile versions of the game. What could be done with a click could also logically be done with your finger on a phone's screen. This allowed the development team to focus less on tuning the movement system and more on everything else further down the line of the development cycle.

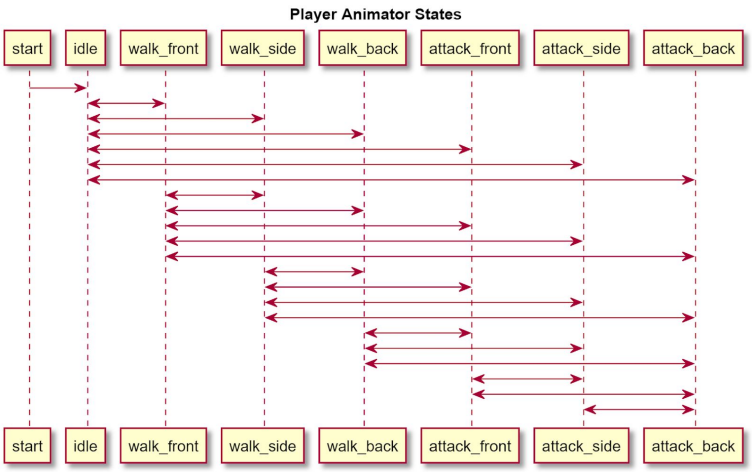
Sprint 2 also involved the creation of some basic rooms that would spawn into the process. Graphics for the floor tiles, doors and walls were created as pixel art and implemented into the game.

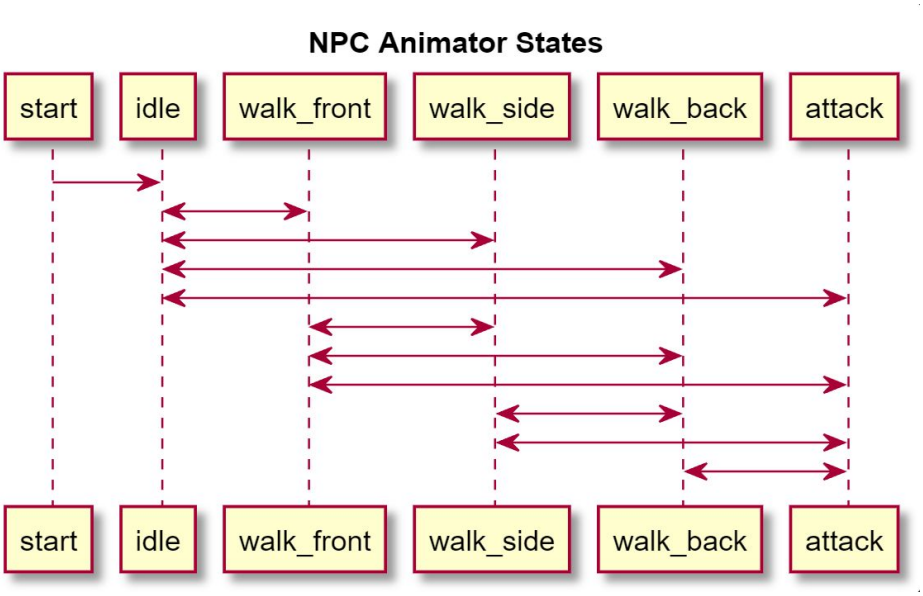


Sprint 3 implemented the health system system into Touch Crawler as well as death. Each entity including enemies, and the player character were given a health attribute. When an acting game object has its health reach 0, it is destroyed. When an enemy object dies, it will drop an item such as another weapon for the player to pick up, or a heart for the player to replenish their health meter. When the player dies, they are sent to the game over screen, shown their score, and offered to start the game over again. Another feature added was “juice” which is the shaking of the screen whenever a weapon or projectile is fired by the player, or an impact occurs within the game. This gives the player a feeling that their weapon has real power to it, and is a key part of incentivizing players to keep playing any video game.

Sprint 4 is when a basic playable build of Touch Crawler was first finalized. A full play session was first tested here. The weapon system was fully implemented. Pixel art sprites were added into the game and put into template rooms. Random procedural generation was fully implemented during this sprint. Rooms were built in which objects were laid about on the ground. Each room had its own unique layout with different sprites, including beds, rocks, bones, torches, boxes and other items. Whenever the player entered into another room from the one they generated, the new room they entered would be randomly selected from the list of templates created by the development team. However, if a player decided to go back to a room they had already been to, they would find the same room that they left previously. Another feature implemented in this sprint was that the health, running score and weapon inventory of the player character was maintained if they defeated a boss and advanced to a subsequent level. Here is a screenshot of Touch Crawler in action featuring some pixel art sprites that were designed.

In Sprint 5, the player was given the ability to pick up weapons laying on the floor and add them to their inventory. Similarly, they also gained the ability to drop weapons they do not desire to keep from their inventory by touching them. It is also where the score feature was implemented. The player would get a game over screen and then see their score and the local high scores upon dying, and be given the option to go back to the beginning of the game. Other small details were added such as a health indicator, particles on weapons, and having a variety of enemies to fight. The movement possibilities for the enemies and the player were finalized, seen in these flowcharts.





Sprint 6 was the final sprint of Touch Crawler's development. Finishing touches were added to the game, finalizing it as a project. Boss fights were added, appearing as soon as the player clears a floor of enemies. Subsequent floors with ever-increasing difficulty were added to the game as well. Whenever the player character defeats a boss, they are sent to another level with more rooms, more enemies per room, and tougher enemies. Sound effects were added for the sounds of weapons firing, the sounds of opening and closing doors, and eerie background music. These served to give the game a more fitting atmosphere, adding a feeling of claustrophobia and nervousness. One of the final features added to the game was that enemies were all given animations for predictable patterns of behavior before they attacked. This is a subtle but important feature of video games that marks a difference between reasonable difficulty and frustrating difficulty. When implemented, these predictable patterns allow the player to learn from mistakes rather than be subjected to death or damage at random.

Results

Touch Crawler's development team set out to create a roguelike dungeon crawler video game that used a similar control scheme for both the PC and mobile versions of the game. As development progressed, it became clear that the design philosophy of the game made it easier to develop than a standard multiplatform game. Rather than having to code for multiple types of control schemes, both the mobile and the PC versions had very similar control schemes, which greatly reduced the time spent on coding two different versions. When the game was finished, it became clear that the control scheme was a benefit for the game. The play testers in the development team enjoyed the usage of their mouse rather than their keyboard for playing and controlling the game. However, they noted that different players may prefer the standard control scheme of PC games which is to use the arrow keys or WASD for movement of the player character and the mouse for other functions.

Conclusion

In conclusion, Touch Crawler's development cycle proved challenging but fulfilling. I came to the realization that it is likely the reason mainstream developers tend to use different control schemes for the mobile and PC versions of roguelikes is that players are used to the standard control scheme of PC games. However, Touch Crawler's development showed that PC based video games should not only be controlled by mouse. There exists an opportunity in the roguelike subgenre to standardize a mixed control scheme where players are offered arrow key and mouse support. The development of Touch Crawler taught me the importance of planning, teamwork and decision making in game development and any other major projects I may work on. I will be sure to apply the lessons learned to any programming endeavors I will face in the future.

Works Cited

Garzia, Andre. *Roguelike Development With Javascript: Build and Publish Roguelike Genre Games with Javascript... and Phaser*. Apress, 2020.

Parker, Rob. "The Culture of Permadeath: Roguelikes and Terror Management Theory." *Journal of Gaming & Virtual Worlds*, vol. 9, no. 2, June 2017, pp. 123–141. *EBSCOhost*, doi:10.1386/jgvw.9.2.123_1.

Ho, Xavier, and Marcus Carter. *Roguelike Ancestry Network Visualisation: Insights from the Roguelike Community*., University of Sydney Australia, 2019.

Halpern, Jared. *Developing 2D Games with Unity: Independent Game Programming with C#*. Apress, 2019.

Takoordyal, Kishan. *Beginning Unity Android Game Development: from Beginner to Pro*. Apress, 2020.

Bouquin, Daina R. "GitHub." *Journal of the Medical Library Association : JMLA* vol. 103,3 (2015): 166–167. doi:10.3163/1536-5050.103.3.019

Tsitoara, Mariot. *Beginning Git and GitHub: a Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Apress, 2020.