The Magic Hat is a 2D asymmetric co-op game where an unlikely duo, a squirrel and a hedgehog, must return a wizard’s hat. You must shoot, climb, and reflect your way through the evil bosses in the forest in order to return the hat to its rightful owner.

Our initial idea actually ended up being the game we decided to implement. However, it was not clear initially who our themed characters would be. Our group was able to come up with a general concept for the game: a 2D asymmetric co-op boss-rush action-platformer with an attacker and defender. During some playtest sessions we were unsure if our idea was good enough to build a game around - multiple playtesters informed us that playing as the defender was boring. We decided to stick with the concept through our 7-8 week development cycle, almost constantly buffing the defender in hopes of making their gameplay more engaging.

Initially, we just had two cubes representing the playable characters, and a blocked-out mockup for one of the bosses (just a moving construct with block hands, and a basic slam attack). About a week or 2 in, we decided to have a hedgehog, squirrel, and platypus as the main characters, adopting a sillier, fantasy theme.

The first weekend of the project (p3 registration), we met in BBB 1695 in order to begin generating ideas. Each person talked about their genre interests, and some even had rough project ideas and simple pitches for games they wanted to build. We compiled all of our ideas on a whiteboard, then brainstormed some game pitches incorporating those ideas. One concept that everyone was interested in was the idea of asymmetric co-op. Everyone thought that this sort of idea would be very cool to implement and would allow for fun and challenging gameplay.

We then thought of a shield and projectile mechanic, where the shield player would be in a defensive role, while the projectile character would be offensive. Both players would have certain abilities that would complement each other; moreover, both players would have to combine their abilities in order to complete the game. We were also fond of the idea of having a “boss hell” in our game, where the game loop would essentially revolve around defeating challenging bosses at each level. We all liked both ideas, and decided to combine the two. We decided to make it 2D as this would be simpler to implement than a 3D game (easier physics and math calculations, reduced complexity).

Our goal was to create a challenging game, where two players would have to cooperate to clear levels. With the asymmetric co-op feature, we wanted to players who had abilities that made them weak on their own. However, when both players combine their abilities in collaboration, they can beat the bosses, and accomplish objectives more easily. We hoped to balance the gameplay such that each player had a decent amount of power on their own, but had greater capabilities when they coordinated their attacks and movements. We also wanted to have multiple bosses (initially, we planned for 10 or more), each with unique abilities that the players would have to work together to overcome. Each boss would require both players to combine their abilities (shield reflections, shield jumping by firing projectiles at the shield, shooting targets) in order to overcome them. Overall, we believe that we were able to achieve this; although the scope of our game shrank over time (from 10 bosses to 3 well-developed ones).

Our process was 7-8 weeks long; we set up 3 meetings weekly before each deliverable deadline. One meeting was on Thursday and 30 minutes long. For these short meetings, each person would go over what they were working on, and what their plans were for the following monday deadline. We talked over each others ideas, and suggested high level ideas for implementing them. Later on, these meetings would be reserved for talking through the previous deliverable's instructor feedback + play session feedback. We discussed possible ways to respond to the criticism that we received, and ironed out changes necessary to our game's design. The remaining two meetings were 4 hour work sessions every Saturday and Sunday. Since we had a 5 person team, it was difficult to get times where all 5 of us would be available. We made these longer sessions flexible so that teammebers could drop in anytime they liked and work together to implement our games mechanics.

We kept a constant stream of communication via these meetings and through our team groupme, always notifying each other whenever we ran into issues. We also gradually set up a system for handling git commits to our codebase: whenever someone committed code, they would have to make a git branch and push that code there. In order to send to master (our ultimate/final official repository), that person would have to make a pull request in order to merge their feature. This system allowed for other team members to review the commit code, and ensure that the code would not conflict or cause issues with the master branch. A "thumbs up" would then be given if the code was ok, and the developer would then be allowed to merge their feature into master. We believe that this system made our 5 person workflow much more manageable and made it less likely for our game to have crippling issues in the deliverable submission.

We heavily utilized jira to create tasks for each member to complete. These tasks would be small and manageable to achieve (such as getting basic movement working, or implementing a simple singleton that could manage the game). Each task, when implemented, would gradually build our game into what it was at the end of p3 gold. We broke our complex design into small, easy to manage, pieces/mechanics for everyone to implement. We made sprints for every deliverable (p3 gold, p3 polish), so everyone had to implement multiple mechanics and bug fixes to the game every week.

After every playtest, we would analyze our game from the playtesters' perspective using their feedback. Sometimes we would receive very crucial information, such as one player not having enough to do while the other player clears the entire level. While other times we received very split feedback, such as preferences for where the jump button should be. After thoroughly analyzing our game with these comments in mind we would sit together to think of changes that we could potentially make to our design to address the concerns our playtesters had. Some of the most crucial changes we made include limiting aim to 32 directions, adding rolling and climbing abilities, and including a close range attack for the defender. These changes to our designs were then implemented before the next playtest session and the cycle repeated.

To do playtesting, we had multiple opportunities. We took advantage of the mandatory monday play sessions to get our game tested with our peers. Our peers, also being game developers, gave us invaluable feedback on the design of our game, the fun behind our game mechanics, and suggested improvements that we could implement to make the gameplay better.

One of the most crucial pieces of feedback we received was in regards to how powerful the players felt as either the squirrel or the hedgehog. Initially, many of the hedgehog players felt worthless and not necessary to beating the golem boss. The squirrel player was able to kill the golem boss without depending on the hedgehog. Even though the shield reflection mechanic was present, it did not help at all and actually made the fight harder to complete. We were eventually able to resolve these issues by redesigning the levels so that the two players would be forced to work together. We also gave the hedgehog a defensive attack which gave him more power.

We also found opportunities to play test our game outside of the normal lecture time. We participated in the play test assignment, and had another studio in 494 test our game with a greater diversity of students (students not in 494, bystanders). This also gave us useful feedback on our game mechanics, and the video footage from those sessions helped us get a sense of how our users were actually enjoying the game.

Our game kept the initial core mechanics that we wanted it to incorporate (shield jumping and reflecting projectiles), but we did end up adding some extra mechanics after some feedback from players complaining about not having enough to do.

However, the biggest changes were to our level designs, our initial boss, the golem, went through several changes as we added an invulnerable jaw to protect its weak point and several stages to make fighting him more exhilarating. A similar change happened to our blazesnake boss as we added more noticeable fireballs and bigger platforms to prevent players from beating the boss unintentionally.

Our team was able to put together a plan and vision very quickly whenever we would meet to discuss big decisions. The team was also able to implement many changes quickly as we received more and more feedback. We felt that the game became gradually better with each play session, and significant overhauls in our design was not necessary.

By using git, we were also able to better manage everyone's work without too many hassles. Although we frequently ran into merge conflicts, git allowed for us to manage code across 5 team members without our main game breaking.

Our team spent too much time worrying about the mapping of the controls when we first started out and wasted quite a bit of time changing the controls before the first couple weeks of playtesting, until we realized that it doesn't matter how we map the controls, someone will always have some complaint about the controls. Another issue that we ran into was time restrictions. We had originally wanted to implement several boss battles, but every playtest session would reveal an issue with a current boss and our team would spend the rest of the week debugging and adding polish to the current bosses.

In the future, we will better prioritize our tasks so that we do not spend too much time worrying about a single feature. Rather, we will consider each change holistically, and prioritize which tasks are crucial for our game's next deliverable, and which ones should be postponed until a later date. In the future, we should consider ways that we can avoid staying up late. There were a few times where some members had to stay up to ensure that a build was submitted before a deliverable deadline. Earlier in our development, many changes were made Sunday night, and sometimes extending into the next day. A better system that would resolve this would be to require any members to submit their code before the end of Saturday, and then build the deliverable submission on Sunday (fixing bugs that come up then). This will reduce the amount of stress we have going into a play sessions as we are not making changes at the last minute.

The most important lesson we learned in EECS 494 is that under a strict deadline, knowing which issues to prioritize is extremely important. We encountered many bugs during development and every week we had to decide which issues were more important to focus on or if it would be better to prototype new features with the time we had.