Final Catapult Report – Team 13 – The Unprofessionals
Medieval Bashout

Richard Caccamise, Tanque Verde High School, Tucson, AZ
Nicholas Shircel, Plymouth High, Plymouth, WI
Alexander Trick, American School of Doha, Qatar

23 June 2021

Project Summary: Medieval Bashout, Team 13

Group Members:	Richard Caccamise	Tanque Verde High, Tucson, AZ
	Nicholas Shircel	Plymouth High, Plymouth, WI
	Alexander Trick	American School of Doha, Qatar

Over the past week, our team has developed a game titled Medieval Bashout using the popular programming language Python. To start, we had a few days of instruction in order to understand the basic principles of the language, as well as some time working with the *pygame* library and PyCharm. Then, teams were formed, and began brainstorming ideas for projects. Super Smash Bros. was an immediate source of creativity for our group and we took inspiration from many of its features in our game. Multiplayer functionality is a key part to Super Smash Bros. so our original goal was to have 4 players playing simultaneously across four different computers. However, we quickly realized that we would have to simplify our game significantly in order to finish on time.

After deciding that we would start with a 2-player game using a single keyboard, we split up to tackle different elements of the project. Members of the team began work on the art, including the background images and character sprites, while other members began the coding work. We started by defining a few classes such as *DamageCounter*, which would later control and update each player's health. We then created the *Player1* class, the wizard, and started to define methods such as *draw()* and *hit_by()*. Once a frame for the wizard character was complete, we compiled code to load it into the game and display a character for the first time. We then added functions to allow the character to move left and right. Once all the frames for the wizard character were complete, we immediately ran into a roadblock. There was no way to animate the character with the current method of movement. This was the first time we really had to combine the knowledge of all the members in our group to solve a problem, but certainly

not the last. Once we created additional methods in the *Player1* class for movement and set the frames to be displayed every five frames in the game, we had a working, animated character. We reused a lot of this code in the creation of the *Player2* class, or the knight, as they are fundamentally the same character.

Now that we had two working characters, a background, and damage counters, it was time to have them start interacting. The first task was to create some simple gravity in the game which was quickly completed. However, the task of stopping the characters when they hit platforms was much more complicated. We had to create a *Hitbox* class that would tell our characters to stop falling when they hit one of the hitboxes, which we put around each platform in the game. We continually ran into issues until we completely revamped the class by replacing the *pygame.collidepoint()* with the *pygame.colliderect()* function so that the entire character would be considered instead of a single point. We ran into additional issues with jumping, as you could hold down the key and fly to the top of the screen; animating jumps and slash attacks, as they would only change frames every five times you hit the key; calculating damage; and more, but we were able to solve all of these issues by working together as a team.

Overall, we are extremely proud of what we have accomplished through this process. All of us came in with little to no knowledge of Python and left with a respectable 2-player video game, along with a plethora of knowledge regarding Python. We discovered that working together was often the only way to accomplish difficult tasks while simpler tasks were most efficiently completed individually. This forced us to learn to communicate effectively, both verbally and through comments in the code, or we would waste precious time explaining our logic and process several times. We all agree that this experience has made each of us a more effective team member and enhanced our communication skills.

Introduction

On the first day of Operation Catapult, Dr. Aidoo and the other faculty members encouraged us to step out of our comfort zone and try something we had little to no prior experience with. We all had little experience with Python and knew we would learn a lot. In addition, Dr. Noureddine explained during the initial project demonstration how the Computer Science field has an immense demand for programmers, specifically Python programmers. Not only would we get to create an interesting and fun video game, but we would attain knowledge that could help us greatly in the years to come. On the first day, we started to learn some of the basic principles and terms of the Python language. During the second and third days, we started to utilize the *pygame* library and completed several of the projects that are part of the CSSE120 course. Finally, after completing the last tutorial project on the fourth day, we sat down with our team members to discuss our project. Our team was formed by the simple fact that we were sitting close to each other, but we quickly got comfortable and started to brainstorm ideas. Through this discussion, the rough outline of Medieval Bashout was devised.

Process

The most important part of any game is a unique and captivating theme, so our group spent a great deal of time brainstorming ideas. We knew from the Super Smash Bros. tournament on one of the first nights here at Rose-Hulman that the game was popular, so we looked to that game for creativity. Of course, a key part of Super Smash Bros. is the multiplayer functionality. Our original plan was to create a four-player game, featuring four characters that each had a generic and special attack. We asked Dr. Noureddine if multiplayer across separate devices was possible and while he said it was, he highly encouraged us to develop a simple, entertaining game before we tried implementing the complicated code. After additional contemplation, we decided that a two-player game would be sufficient and they would use the same keyboard to move the characters in the game. With that decided, we moved on to the theme of the game. We found that this was difficult to determine without brainstorming characters first, so we moved on to that. A wizard, or mage, was one of the first mentioned and once it was decided that we wanted to use it, we determined that a medieval theme would be appropriate. With this came the idea for the knight character. However, two characters and a theme do not magically create an entertaining game. We decided that having platforms for the characters to jump on was important and the characters would need two attacks each. It was determined that a slash attack would work for both characters, with the wizard also being able to conjure a cone of fire and the knight able to use a grappling hook. As we began to develop the game, however, we realized that we couldn't quite fulfill all of our grand aspirations with our limited time.

The first task we had to accomplish was the initial setup of our project in PyCharm. We created a repository for our code using the GitHub service and added a project file with its associated information. Throughout the creation process, we added sprites, audio tracks, and additional files to this repository for easy access. We also took advantage of an application called Git and by linking it to our GitHub repository in addition to the PyCharm application, we were all able to work together on the same code. With PyCharm ready to go, it was time to start coding. We started by running the program to confirm the base code that was constructed was already valid. We proceeded to create the *DamageCounter, Player1, Player2, KnightSlash,* and

WizardSlash classes. For each class, we created an *__init__*, or initialize, method and passed through variables such as *screen, x, y,* and *font_color.*

```
class DamageCounter:
    def __init__(self, screen, x, y, font_color, player):
        """Gives the base functions for the damage counter that will display the health of the characters"""
        self.screen = screen
        self.x = x
        self.y = y
        self.health = 100  # this value will be mutated throughout the game
        self.font = pygame.font.SysFont("", 40, False, True)
        self.font_color = font_color  # must enter an RGB value when creating an instance
        self.player = "Player " + str(player) + ": "
```

The image above shows the code for the *__init__* method for the *DamageCounter* class. You can see that for every variable that needs to be passed through, such as *screen*, there is a function that does so. There are also additional functions such as *font* that are not passed through but need to be referenced by other functions or methods within the class. While working in a class, you are unable to reference anything from other classes, which often leads to repetitive creation of similar functions and methods. After the basic functions for each class were laid out, we moved on to refining the *DamageCounter* class.

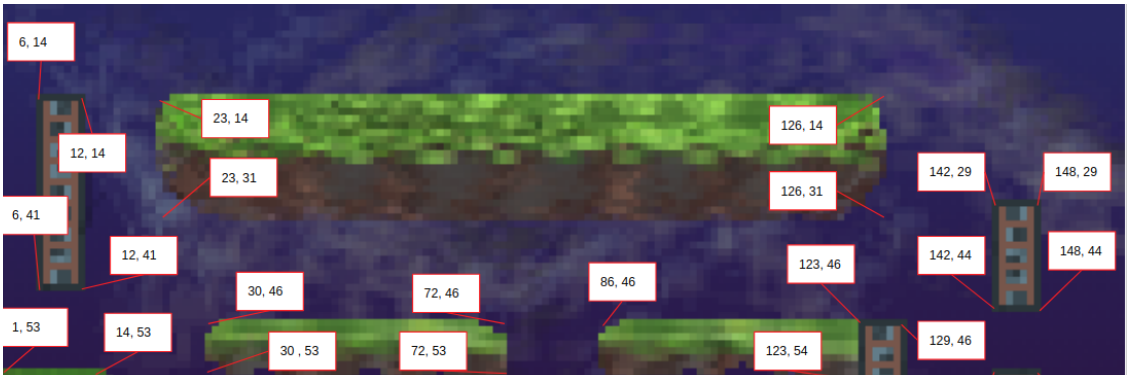
During this time, members of our team made significant progress on the graphics for the project. We split the art design process into 2 parts; character and attack design along with background and logo design. Richard did the art for the background and logo while Alex did the art for the characters, attacks, and title/ending screen. Unlike a game engine, the animations for the attacks and characters could not just be imported into the code. Instead, we had to draw each frame of the animation, and then import each frame as a different image. This process was completed fully by hand, and was a big limiting factor in terms of character designs and time constraints. Each character was inspired by basic fantasy stories, and other art found online, and were also given a darker border around their light bodies to make them mesh better with the darker background. For the projectile attacks decided on late in the creation process, it was decided that we would use one single image, instead of more frames to conserve time. The beginning and ending screens were simplistic, but fit the goals of what they needed to do well enough. Richard, during his time, started off with sketches of a stage design while taking into account how accessible that stage would be for the characters. Once a sketch was decided on by the whole team, he used a pixel art program to draw the basic outline of the stage to correctly map everything. After he thought the basic outline was good, he drew the final stage and background, along with labeling all the hitboxes. Going off of the theme of the game, the stage has magical floating islands in front of a purple sky and a mountain. For the logo design, Richard wanted something formal yet unprofessional to fit our team name. This immediately made him think of Skipper the Penguin (an unprofessional penguin) with a formal tie, so he went with that.

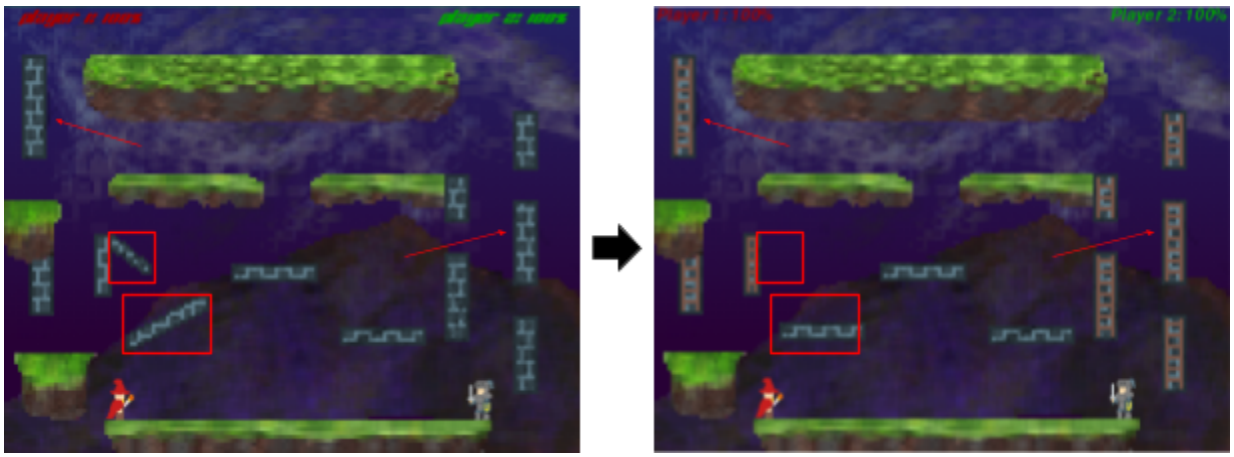
Now that there were graphic files to reference, it was much easier to create additional methods for the *Player1* and *Player2* classes. We decided to focus on the *Player1* class first. We used the *pygame* library to easily load our wizard sprite into the game and then displayed it by blitting it onto the screen. We continued to add additional functionality by allowing the wizard character to move left and right across the screen using the arrow keys. We also implemented jumping, but we had yet to create any gravity to pull the character back down to the ground. In preparation for future collision detection, we added a hitbox to the *Player1* class and tested it out

with a quick mouse-click function. While implementing this test, we also created *damage_counter1* to control and keep track of the wizard's health.

At this point, the frames for the wizard were complete and it was time to implement animation. With help from Dr. Noureddine, we were able to fill gaps in our knowledge that would have prevented us from the implementation of this animation. We started by changing sprite images according to the game's framerate, which is 60 frames per second, but that created an animation that was too fast to see. We ended up cycling images every 5 frames to mitigate this issue.

The creation of the knight character followed and was completed quickly now that we had the necessary knowledge. We then implemented some gravity code for both characters with the help of our faculty mentor. At this point, we decided to take on the challenge of creating all the hitboxes and developing collision detection code. Using a formula we developed, we were able to input coordinates straight from the background image and convert them into the appropriate coordinates for the game. We also created a *Hitbox* class to aid in the creation of the hitboxes.





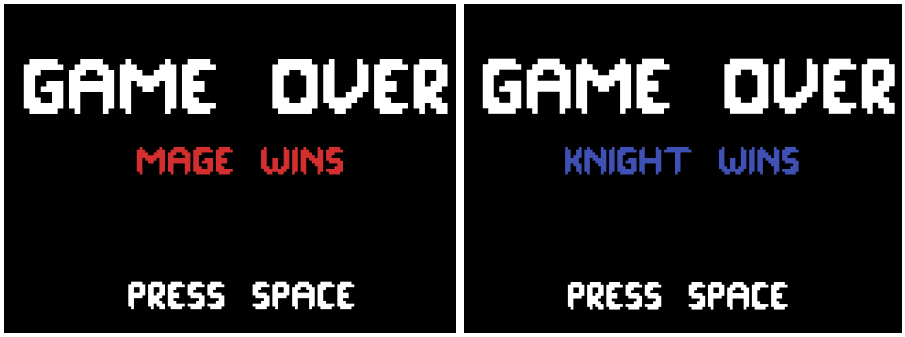
In the images above, you can see a portion of the document we used to log coordinates along with the evolution of our stage design. We found that it was too difficult to create hitboxes on a slope, so we were forced to iterate on our stage design to remove that complication. We also decided to simplify the game slightly and include ladders instead of implementing a wall-jumping method. This gave us additional time to work out the bugs with the hitboxes and other elements of the game.

Now that there were two almost fully-functioning characters, it was time to write some more classes. We proceeded to create the *KnightSlash, WizardSlash, Fireball,* and *Arrow* classes. We had decided earlier on that it would be easier to create a fireball and arrow than the attacks we brainstormed earlier. The animations gave us some trouble again, but we persevered through the issues. When we were ready to start attaching damage to each attack, we again realized that a hitbox for each attack was necessary to tell if the attack was hitting anything. At first, the projectiles would kill a character instantly, so we had to create a method to delete the projectiles when they collided with a character. We also took this time to limit the amount of projectiles allowed to three per character. Once a projectile leaves the screen or collides with a character, they will be able to fire another.

With fully functioning attacks and movement complete, there were only a few things left. A background music track was found for the game along with sound effects for slash attacks, jumps, fireballs, and more. We were surprised by how much these sound effects improved the overall experience while playing the game.

We then spent several hours working through various bugs in the game and implemented some additional, small features that improved the overall game. Improving the jump and slash animations took a good portion of this time.

Some of the final implementations were the intro and ending screens. Up until now, the game started right away when it was opened. Members of our team did some additional artwork to create a title screen that is featured on the cover of this report. This allows the players to get ready before one of them presses Enter to start the game. The game over screens shown below are displayed when the health of either character reaches zero. For example, if the wizard hits the knight with ten fireballs, the knight falls off the screen, or various other scenarios, the knight's health will be at zero and the "Mage Wins" ending screen will be shown.



Finally, our group added comments throughout the program to give clarity and explanation to our code for anyone else that may look at it.

Challenges Faced

Throughout our time coding and drawing artwork, we ran into several challenges and roadblocks to our final goals, which detracted from our time, yet gave us greater debugging experience overall. First, as we decided on doing pixel art for our artstyle, it became a bit of a challenge when we attempted to import a 32 x 48 pixel character onto a 960 x 720 screen. The characters appeared, but were incredibly small. The same happened with the background, and anything else we attempted to port in. Luckily, this was a relatively simple fix, as *pygame* already has a scale method, and so not too much time was wasted over this conflict.

After we had imported the photos and added the hitboxes and movement, another problem arrived; the players could fly using their jump method. Things weren't rendered with full physics, and so the "jump" was more of an upward teleport, meaning the floor wasn't required. Our solution was to make it required, as we added a boolean that gets turned off once they jump. When they return to a platform, it is turned back on. This disabled the flying from before, but still enabled a midair jump for recovery if they fall. However, our solution created another problem. Now that we only allowed the character to jump once, we couldn't have the key press functions in the main game loop. This rendered our animation code useless and it really did look like the character was teleporting up instead of jumping. After a great deal of contemplation, we discovered that by splitting the jump into smaller jumps with frames rendering at those intervals, we could make a simple jump animation.

Surprisingly, many of our big problems came from the title screen itself, including putting it up in the first place. We struggled for a bit, but decided to make it the initial screen which then switches to the stage on a button press. While this did work, it created a second problem; the characters still spawned in. They could move, jump, and even die before the game began. This was fixed by both disabling all movement from them, even gravity, till the button was pressed, and moving them upwards so they looked as if they were part of the title screen. And then we ran into the hardest challenge, the game's name. We had no idea for the longest time, until we decided to keep with the placeholder "Medieval Bashout ", after we fixed an initial misspelling.

Group Project Reflection

Over the course of the last week and a half, we have all learned how to work together as an effective team. Throughout the brainstorming and development process, we had to use excellent communication to convey our thoughts and ideas to the rest of the team. We learned that setting reasonable expectations and goals was important to making progress and not wearing the team out. We each have varied backgrounds and therefore have different opinions, thoughts, and methods of doing things. While this can be an issue, it only helped our group solve problems faster than we could have individually. Problem solving is an essential skill in any field, but it is absolutely crucial in the field of computer science. While each of us solved problems on our own, it often took the perspectives of several team members to solve a complex problem effectively, and also efficiently. With our time constraints, limiting our time spent on specific elements of the game was vital. Each day, we came with a good idea of what needed to be accomplished even if we weren't quite sure how. Whenever we finished a task, we would ask the

other members if there was something they needed help with or another task they had in mind. We also constantly communicated regarding the condition of the code and whether or not we had the most up-to-date version. Each of us accomplished a great deal individually, but without the cooperation and effort of each member, our game would not be what it is today.

Individual Project Reflections

Richard Caccamise

I had very little experience with Python before attending Operation Catapult. I took Software and App Design in 10th grade. We were taught Python for two months before COVID shut down the school. In two weeks at Operation Catapult, I learned more about Python than I did in two months at my school. Not only did Operation Catapult teach me a lot about Python, but it taught me how to code with a team. Each team member had a different strength. We were able to connect our strengths to help each other out, fix bugs, and come up with ideas. If one person had a hard time understanding a part of the code, another person would help explain what's going on. If one person had a hard time fixing a bug, another person would help them fix it. I specialized in background design, fixing bugs, implementing the title/win screen, preventing spamming, improving game mechanics, helping each other out, and logo design. Nick specialized in coding the classes, implementing game mechanics, improving game mechanics, helping each other out, hitboxes, and fixing bugs. Alex specialized in character design, title and winning screen design, the game's theme, weapon and attack implementation, sound design, animations, fixing bugs, improving game mechanics, and helping each other out. At every step of the way, we communicated. We shared our ideas with each other, asked for feedback on our ideas, created a plan for the whole project, communicated who was doing what, asked each other what needed to be worked on, made sure others were okay with you adding something, asked each other for help, and asked others if they needed help. I loved working with Nick and Alex, they were awesome team members and I hope I can work with them again.

Nicholas Shircel

The extent of my experience with Python before coming to Rose-Hulman was what I could absorb through a one-hour YouTube tutorial. I had done some basic work with JavaScript and Arduino, but we bypassed my skills in any programming language after the first day. One thing that I had never experienced before was working on a coding project with a team. I have always been one to object to group projects and I am usually more comfortable working on my own. However, the last week and a half have proven to me that groups are not always handicaps, in fact they can be the very opposite. The combined knowledge and experience of the members in my team has propelled our video game past what I could have imagined. To be honest, I thought Alex and Richard were extraordinarily ambitious during the brainstorming process, but their drive and creativity has resulted in something truly amazing. Alex did a fantastic job creating the characters and attacks while Richard worked hard to create an amazing background for the game. We all had moments of frustration and times where we would stare blankly at the screen, but the combined effort of the team always fixed the problem. With more time, I guarantee we would have solved even more of the problems we faced and created an even more intricate and complex game. I know the other Catapult participants and our parents will be blown away by what we have accomplished as a team.

Alexander Trick

When I went into this, I had a very bad feeling about how things would go. I entered with little to no python knowledge (I had a good grasp on Java, which helped in the end), and I also had a bad track record with team projects. In the sense of fighting more with my team than with the challenge. And I can joyfully say that I was worried about nothing. Richard and Nick were wonderful teammates, who did fantastic jobs. Nick did a massive amount of the code, and led the team effort in that direction as Richard and I worked on the game's art and music. Richard, while speaking a bit too much on Pepsi-Man in my opinion, gave great insights into fixes and improvements we could make to the game. As a team, I think we were a bit too ambitious however, we wanted to have online multiplayer, mouse control, 4 characters. It was all too much at the time, and now with having had to fight python for the past week and a half so that it would do what we wanted, it feels silly now about how much more we wanted to do. Overall, it was a good experience, and I hope to work with them again in the future.

Conclusion

In summary, our team has accomplished a great deal in the past week and a half. We started by building our individual knowledge of Python and then worked together to learn even more. There is no denying that we all have varying levels of experience and different perspectives, but we used this varied knowledge as a team to accomplish great things. While sitting in front of a computer for several hours a day can be tiring at times, our team persevered and continued working hard throughout Operation Catapult, even taking extra time in the evening to work on aspects of the project. We took on the challenge of coding such an ambitious game without missing a step and finished with a total of 1000 lines of code. We hope to have more opportunities to code and learn more about computer science as we finish our time in high school and move onto college.