Introduction

STEM-related fields are growing in response to both the rapid development and demand of technology. This is made even more apparent in the modern day by the pandemic and need for work-from-home (WFH) environments. However, it remains intimidating for children to pursue an interest or career in such a field, especially when STEM topics are stereotyped as challenging or difficult. Furthermore, STEM unfortunately continues to appear reserved for only the most privileged members of society.

The main goal of Seeds in STEM is to raise students' interest in STEM fields, regardless of their background or previous conceptions of STEM careers. This specific project, M³, aims to facilitate the introduction of kinematics to younger students through an interactive and fun video game that is coincidentally made possible through a STEM class at Cal Poly SLO.

Currently, there are many educational games available for free play. However, most of these educational games focus more on the “education” rather than the “game”, giving players the impression that they are slogging through a lesson plan. M³ seeks to rectify that shortcoming by prioritizing fun and meaningful play, with educational lessons as an optional feature for players to engage in if they are interested. However, players will still have to be mindful of basic physics, such as mass, velocity, and friction, in order to clear all of the obstacles in their way.

Unfortunately, since “fun” is a relative term, it will be difficult to ensure the quality and consistency of gameplay for all players. In an attempt to rectify that, M³ has gone through an iterative design process and multiple playtesting sessions to receive feedback and suggestions from a variety of users.
Application

M³ is a puzzle-platformer game that takes place in a modern, alternate universe. Users play as a space ranger who was forced to land on an asteroid after their team ran out of fuel on their voyage to the planet Hydeq. Three types of fuel crystals have grown in random and oddly hard-to-reach places around the asteroid headquarters. The journey through the asteroid features different types of crystals and various ways to obtain them.

![Types of fuel crystals]

Players must use their developing knowledge of physics in order to collect enough fuel crystals to power up their ship. Only then can they escape the asteroid and continue their voyage to Hydeq.

We have published our educational physics game on Unity Play, an online platform for game developers to share their creations and works in progress. By going to the following link, users can play immediately after the game loads. We recommend enabling audio and fullscreen, as shown in Figure 2, to enjoy the full effects of the game.
The controls are described in the “How To Play” window, reachable from pressing the “escape” key.

Players also must work through an interactive tutorial level that teaches them the controls and basic game mechanics through the background graphics.
The controls are summarized here as well:

- ‘A’ to move left
- ‘D’ to move right
- Spacebar to jump
- ‘F’ to pick up objects
  - ‘R’ to select a specific object to pick up (if multiple are around)
- ‘E’ to launch objects
  - Hold ‘E’ down to adjust the initial velocity of the object
- ‘W’ to rotate the launcher canon up
- ‘S’ to rotate the launcher canon down
- Escape key to pause the game
- 1-9, 0, -, = keys to select respective objects in the inventory
  - Additionally, you can use the mouse scroll wheel to scroll up or down the inventory slots.

M³ can be found at this link: [game link](#).
The trailer for M³ can be viewed at this link: [trailer link](#).
Background

This senior project is a part of the larger Seeds In STEM program. Seeds In STEM is a two-part program, Generation STEM and Full STEAM Ahead, which provides STEAM education for K-12 students in the Santa Maria community in an inclusive, sustainable and impactful way. Due to the large scope of this program, there are many specialists working together to bring this vision to reality.

As a result, there are various teams working on specifics such as 3D modeling or storyboard drafting, allowing the software developers for this project to focus on game design, development, and implementation in Unity.

Unity is a real-time game engine, capable of rendering physically correct materials and lights in the editor. The game engine enables users to create games and experiences in both 2D and 3D through custom scripting in C#, plugins, and drag-and-drop functionality. The advantages of using Unity include: the prefab system which makes reusing code and assets from other projects easy, the expansive library of resources for all creators to reference, and cloud-based tools to monetize one’s project and include multiplayer options. Unity also has a large community of creators who are able to facilitate each other's learning in a collaborative and accepting environment.
As with any game engine, there are also trade-offs that might make it less appealing than other engines. Unity includes many features that some users will never find relevant, making it more cluttered than necessary. This also means that the Unity engine is less specialized than other engines that focus on one type of design, like 2D games, lighting, or rendering. Lastly, creating a blank 3D project alone requires more space than other game engines, which can make Unity unsuitable for some machines. [1]

Design

The main goal of Seeds in STEM is to make STEM topics, such as kinematics and engineering, interesting for Santa Maria middle school and high school students to spark their interest in such topics later in their educational career. Specifically, our senior project falls under Full STEAM Ahead, which is a day where TRIO students can come to Cal Poly to learn about different engineering disciplines and organizations on campus, in order to inspire students to pursue engineering and see the different disciplines that fall in STEM. The goal of our game in the scope of the larger Seeds In STEM program is to make learning STEM material fun, easy, and intriguing to middle school students in the Santa Maria area.

With these goals in mind (fun, easy, and interesting), our game design falls upon making them a reality.

Fun
Meaningful play is one such aspect in achieving a fun game, and was one concept we were most conscious of while drafting the initial game mechanics. Naturally, if a player feels their choices affect the outcome, they will have a greater interest in, and become more immersed in, the game and what it has to offer.

In $M^3$, players (also known as space rangers in the game’s setting) explore each level to achieve the subgoal of collecting fuel crystals. Within each level, they are able to play at their own pace and observe their own actions upon the stage. These
observations can be, but are not limited to, launching different types of projectiles, attempting to climb items shot into their own desired arrangement, and viewing the reactions different fuel crystals have when hit with different items.

To further elaborate on the gameplay, the main challenge of M³ is mastering the core mechanic: picking up and launching items. As a result, players must be mindful of aiming the launcher by adjusting angle and velocity of their shots, the item they are launching, where they position themselves prior to shooting, and at what they are shooting. Every object launched is given an initial velocity rather than a force. We decided on allowing the players to manipulate the initial velocity, rather than force, to emphasise the underlying concept of how each object travels the same path regardless of mass (if given the same angle and velocity). Launching projectiles with an initial velocity also eases the learning curve at the start of the game by removing force-based calculations a player would have to make otherwise. Furthermore, the game’s two-dimensional layout allows players to solely focus on the x- and y-components of their decisions.

The core mechanic naturally leads to a variety of choices the player can make. Most notably, within each level, space rangers will face a situational choice of which item is most efficient to use when clearing individual obstacles and collecting crystals. For example, a previous level’s solution of using standard cubes and spheres as projectiles may be insufficient in a level that requires the use of bouncy balls. Likewise, bouncy materials may be less ideal to use when players want projectiles to remain still (for example, when players want to climb their man-made vantage points to target at a fuel crystal).

Players are also given the choice for the order they wish to collect crystals in (barring the tutorial level) and the order that they shoot items, leading to non-linear level design. In order to proceed to the next level, the space ranger must collect (or attempt to collect) all possible fuel crystals on the field. A level is considered “complete” when all fuel crystals are collected. In the case that a player wants to get a faster time or play more efficiently, they are able to retry the level at any time. This gives the player a sense of mastery and some inherent replayability as the player learns of the game’s mechanics and new strategies for progressing emerge.
Our game also contains a hidden positive feedback loop inherent with the core mechanic's design. Players who use their items more efficiently early in a level are able to make more mistakes without having to backtrack. As the levels continuously provide enough items located at each puzzle to progress, excess items from previous puzzles act as additional tries and reinforce players' choice to keep more items at hand.

For completing challenges and puzzles, players are given various rewards. Typically, players are given rewards of access, allowing them to obtain items of various shapes, masses, and properties. Likewise, with increased access to the level, they are able to explore more and collect crystals that were once restricted. Upon completing a level, players are given a reward of glory in the form of stars and the time it took them to complete the level. This encourages the player to improve by learning the design of each level and earn greater rewards.

Each level contains a mixture of perfect and imperfect information. Prior to the start of the level, there is a quick camera pan across the whole level, allowing players to know the general layout of the level before playing. Likewise, at a glance, players can easily determine the total amount of obtainable fuel crystals. However, what they do not know is where each fuel crystal is, what the solution to obtain each fuel crystal is, or the available items at their disposal. On the other hand, all information about an item in their inventory is made transparent: players can see the mass, name, and additional physical properties (such as frictionless or bouncy materials) of the object. This adjusts the cognitive flow and allows the player to make informed decisions about where, and at what, they are shooting the item. It also helps them distinguish between similarly looking items with information that would otherwise be unavailable.

Lastly, at a more surface-level of the game's design, we aimed to reward the player for engaging in even the most primitive actions. For example, when the space ranger is walking, jumping, shooting items, or collecting fuel crystals, there are sound effects accompanying the action. These sound effects aid the player in interpreting the scene as part of invisible design.
The sound effects are intended to be pleasing to act as both a reward and positive reinforcement for interacting with what the game has to offer. One example of this is the sound effect for collecting a crystal. Based on the same concept that the secret item sound from the *Legend of Zelda* games provide, this sound aims to provide a reward invisibly without inherently interrupting gameplay to give it to the player. In the same vein of conditioning, if the player errs, such as breaking a fragile crystal, the accompanying visual and sound effects are designed to notify the player of their mistake.

**Easy**

Another driving force in M³’s design is the game's intended audience: a rather large range of ages; from middle schoolers to high schoolers. Moreover, the game is intended to be played as supplemental material for the Seeds In STEM workshop. With these two factors in mind, we were conscious of each player’s skill level and decided we did not want to frustrate players with difficult gameplay.

Most notably about easing the game's difficulty is the “Next Level” button in each stage. The button, true to its name, transports the player to the next level, regardless of their progress in the current level. While it’s a unique mechanic to a puzzle game, as it allows players to trivialize the puzzles in said game, it was deemed a positive change. We did not want players to be locked out of levels because they could not fully clear a level prior.

The “Next Level” feature naturally affected the game's individual level design (barring levels that introduce a new mechanic). Each level's fuel crystals can be broken down into two categories: crystals that are obtained only after clearing a puzzle (that is affected by external factors such as ice, wind, or both), or crystals that typically can be shot at directly (but other factors, such as the angle, still must be considered). As expected, these puzzle fuel crystals are designed to be more difficult than the latter crystals. With the “Next Level” mechanic, players are not punished if they are unable to solve a puzzle. Instead, they can ignore the puzzle crystal, collect the easier ones, and press the “Next Level” button to the next level.
An additional decision in lowering the game's difficulty is the lack of a death mechanic. Likewise, there are no enemies to impede progress. This allows players to explore, solve puzzles, and experiment at their own pace. However, even if there is no fear of death, there is still a checkpoint system. The checkpoint system, while not necessary, is intended to ease the frustration of the player if they make a critical mistake. Instead of fully restarting the level and losing the progress they might have made prior, they can instead reset their position to the latest checkpoint they activated and try again.

Lastly, there is a negative feedback system in place when a Lulu or Airam pop-up appears. These characters appear to give the space ranger hints on how to proceed or of what mistakes should be avoided. In our playtesting, the hint was sufficient in guiding the player's next steps to finding the solution and moving on.
Interesting
We were also deliberate in the setting of M³ to make the game interesting. Besides the guidelines provided by the Seeds in STEM workshop, there are additional embellishments to how the game is presented.

As required by the workshop, M³ is set on an asteroid in a modern, alternative universe. To achieve the feel of a sci-fi space adventure, there are many different visual aspects chosen to match the story elements. For example, in the background, one can easily see both the asteroid's terrain and the stars in the distance. Likewise, the matter launcher looks as one would expect from space adventurers: high tech and self propelled following the player. Furthermore, the space ranger's space ship is sleek like a jet plane, rather than a clunky aerial vehicle we may see today. The fuel crystals, while obviously crystals, still hold an otherworldly vibe with their ethereal glow and unknown composition. As the game progresses, there are also pop-ups of the story's characters, Airam and Lulu, to aid the player in their quest and provide commentary.

Furthermore, we have added many animations to keep players interested. During the beginning scene, players watch the space ranger crash land on the asteroid's surface. Likewise, at the end of the game, we can see the space ranger, Lulu, and Airam celebrate before leaving the asteroid in their sleek spaceship.

In using audio as a mood setter, sound effects are intended to be a mix of both realistic and sci-fi, giving depth to the environment. Background music provides an empty and mysterious feel, which emulates that players are trapped on an asteroid without outside contact, as one would expect while stranded in space.
Implementation

We created several C# classes over the course of our game development. Following, we include an in-depth description of the main classes that make up M^3, including how their interaction with each other contributes to the game’s mechanics.

**Figure 6: Relationship Overview Between Main Game Objects**

**FuelCrystal.cs**
This script describes the basic functionality that fuel crystals and their variations all have in common. The following is a quick overview of the fuel crystal functionality.

**Figure 7: Fuel Crystal Overview**
As shown in Figure 7: Fuel Crystal Overview, the fuel crystal’s rigidbody is first set to `isKinematic`. This prevents it from being affected by forces, such as gravity, and is used primarily to have fuel crystals hover in mid air.

Upon trigger with the player, the crystal is set inactive to simulate its collection. It also plays sound effects, notifies the UI of its collection, and instantiates a temporary particle effect.

Upon collision with an item, the `isKinematic` field is set to false, allowing the crystal to react to the collision (and other forces possibly at play).

**Inventory.cs**
This script is the backbone of the Inventory system. It keeps a list of the items currently in the inventory and updates the UI whenever necessary.

*Figure 8: Inventory Overview*
Upon start, the script obtains references to various UI components, such as Text or Image. This is done to quickly update various fields without having to call GetComponent<>() on every change. The script also handles the following key inputs to highlight/select a specific item from the inventory bar: ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘0’, ‘-’, ‘=’, and alternatively, ‘mouse wheel scroll up/down’.

When the player collects a new item, Inventory.cs adds the item to its list and notifies the UI to add the item’s corresponding image. Furthermore, the Inventory UI slot with the new item flashes a bright color for a few seconds to indicate the player of the new addition.

Whenever an item is removed, Inventory.cs removes the item from its list and notifies the UI to remove the item’s corresponding image.

Upon any change to the currently selected item, either through adding an item, removing an item, or when the user selects a different item to launch, Inventory.cs updates its Item Info Panel property. The Item Info Panel displays information extracted from the corresponding item (name, mass, and additional physical properties, such as frictionless or bounciness).

**Item.cs:**
This script describes custom information about an item. Its public properties, `NAME` and `IMAGE`, are used by the Inventory to populate the Inventory Bar UI and Item Info Panel UI.

It also has the following private variables:

- `outOfBounds` - The maximum distance the item can be from its starting position
- `minSize` - The target size of the item after shrinking. Prevents the item from shrinking to a negative scale.
Upon start, the item’s default attributes, such as its starting position and size, are stored. Furthermore, as the item shrinks upon pick up, we make calculations to determine the amount the item shrinks per frame (also known as shrinkFactor). This shrinkFactor is dependent on both the item’s starting scale, the desired minimum scale after shrinking, and the amount of time allocated for shrinking (default of 0.5 seconds).

During the update, Item.cs handles two mechanics. One is checking whether the item is outOfBounds units away from its startPosition. This occurs on the rare occasion that the item falls off the stage. The second mechanic is the shrinking visual effect. When a player tries to pick up, the item is notified and begins
shrinking. Once the item has shrunk to a certain size, it then notifies the listeners that it is ready to be picked up.

Player.cs:
This script is used to capture user input and play the appropriate sound effects.

Figure 10: Player Overview - Initialization and Update
As the core script for player control, it handles ‘A’, ‘D’, and ‘SPACE’ inputs to make the player move left, right, and jump. With ‘R’ and ‘F’ inputs, the player is able to focus on a nearby item in the stage or pick up the selected item. When the player is idling, moving, or jumping, the animation is set accordingly through calls to its animator. This script also plays the respective sound effect when the player jumps, walks, or picks up an item.

As seen in *Figure 10: Player Overview*, when the player hits the ‘R’ key, they are able to change the focus of the current item to another item in pick-up distance (if applicable). This mechanic is supplemented by the Trigger functions, which populates a list of items that are in range of the player. When ‘R’ is pressed, the currently focussed item is changed to another item in the aforementioned list. Players are able to determine which item is selected prior to picking it up, as the selected item is outlined.

![Figure 11: Outline of selected Item](image)

Player.cs is also a listener to various events, which are used to enable or update various functionality. The following flowchart (*Figure 12: Player Overview - Triggers and Events*) is a quick overview of how the Player class handles triggers and incoming events.
Figure 12: Player Overview - Triggers and Events
PlayerController.cs:
This script is used to maintain relevant background information about the current level. This background information includes: the total possible fuel crystals in the level, the number of fuel crystals the player has obtained or broken, and the total time spent on the level.

PlayerController.cs is also able to recognize when the win condition is achieved. When the win condition is met, or the player clicks on the “Next Level” button, it begins its EndGame() routine. This routine includes (in order): an ending level animation of the Space Ranger celebrating (if all crystals are collected), calculating
the player’s star rating for this particular run, and showing the Level Complete screen upon completion. Star rating is based on the following: time, the number of times the player shot a projectile, and (if applicable to the level) whether the player broke a crystal or not.

**ProjectileLauncher.cs**

This script is the backbone of the matter launcher. It handles user inputs, while maintaining the integrity of the launcher’s rotation. Furthermore, it works closely with the Inventory class, which provides the projectile launcher items to shoot.

![Figure 14: Projectile Launcher Overview](image)

Next, we will briefly describe the role and purpose of several other classes which play a less crucial role in the overall game mechanics.
CameraPan.cs
Given a list of positions, the camera moves from one point to the next. It is used to
give a brief overview of the level's layout before the player begins playing. Upon
completion, it triggers the event “Camera Pan Done” to notify listeners that the tour
is complete and that the game can begin.

CameraShake.cs, RegCameraShake.cs
Shakes the camera when triggered, e.g. when a Fuel Crystal is hit by an Item.
CameraShake applies to Cinemachine Virtual Cameras, while RegCameraShake
applies to the default Unity cameras.

CameraZoom.cs
Subscribes to HypeCrystal's event. When the event occurs, the camera zooms into
the crystal and changes the time scale to slow motion. After a certain time has
passed, the camera zoom effect is removed.

CheckPoint.cs, MasterCheckPoint.cs, PlayerToCheckPoint.cs
The checkpoint system. Stores the position of the latest checkpoint the player has
passed, and brings the player to said checkpoint if demanded.

EndSceneDialogue.cs
Displays an image (in our case: LuLu and Airam) and text (dialogue). Images and
text are pulled from a provided list, and are changed when a developer specified
time has passed. This allows us to string multiple lines of dialogue/expression
changes in a sequential fashion to simulate character dialogue.

FragileCrystal.cs
Drives the logic behind a fuel crystal's “fragility”. It determines when the fuel crystal
is hit, how hard it was hit, and will destroy itself if hit too hard. When destroyed, it
notifies listeners of the event “Fragile Crystal Broke”.
**HypeCrystal.cs**
This identifies which crystals will get a Camera Zoom effect, which occurs only when an Item is within trigger range. Notifies listeners of the event. This event can only occur once per fuel crystal.

**LevelSelect.cs**
Dynamically creates a level select layout based on the number of levels in the game. It detects and conveys the player's progress in the game (ex: a completed level's star rating, whether a level has been completed, etc.) and locks any levels the player is unable to play (due to not completing the level prior).

**PanToCharacter.cs**
Navigator for the possible characters the player may choose from during the character select. It also updates the character's name and backstory depending on the currently selected character.

**RockCrystal.cs**
Drives the logic behind a fuel crystal's “hardness”. It determines when the fuel crystal is hit, how hard it was hit, and will free itself once it has sustained enough force.

**TimelineDirector.cs**
Allows the scene's Timeline object to be started and stopped from another script. This is helpful when the Timeline is triggered by a player action or another event in the scene.

**WindArea.cs**
Applies a force in a given direction to all game objects in a provided area. This is used to simulate wind.
Analysis and Verification

We were able to measure our success through a number of methods that we learned through our game design class. First, we playtested with several users in order to locate the pain points in our game. We playtested a few times throughout development in order to evaluate whether our bug fixes and experience improvements were actually effective. We asked playtesters about their gameplay habits and demographics before they played the game in order to get a sense of the people we were designing for. Then, while the playtesters ran through our game, we took notes on their reactions and habits. Finally, we asked them to fill out a survey after playing the game. Our survey questions included, but were not limited to:

- Did you feel challenged? (scale 1-5)
- Were you able to figure out how to play on your own? (scale 1-5)
- Was anything about the game confusing?
- How did you feel about the game controls? (move, launch, pickup items, etc...)
- What physics concepts did you have to use to make it through the levels?

We were able to improve gameplay experience through these surveys, and saw firsthand a high level of enjoyment from playtesters, even before deployment.

Unfortunately, there is little valuable information that we can pull from Unity Play surrounding the success of M³. Currently, our game has 104 playthroughs, but the majority of these are from team members, classmates, and playtesters running the game. We estimate that at least 30 of these playthroughs are from Seeds in STEM workshop attendees. We also know that the three workshops totalled about 150 attendees; however, it’s unlikely attendees were notified about M³ being released.

M³ is successful when considering scalability, however. No payment, subscription, membership, nor download is required in order to play the game. The only constraint is that players must have access to a computer, since the game is online and requires keyboard input.
The efficiency and speed of our game depends only on the player’s WiFi connection and access to WiFi. However, the game is available to download locally so this limitation can be avoided.

Based on our final rounds of playtesting, we found that users were unable to locate any bugs or inconsistencies in M^3. Our physics components are based on Unity’s physics engine so the accuracy of the physics in our game is only as accurate as Unity’s implementation. Altogether, we believe the final game is accurate and fault-free.

Delving into our post-game survey results, the majority of playtesters were able to understand how to play M^3 without outside assistance. This is encouraging since the game should be able to be played without developer interference if we expect it to be educationally effective. Following are screenshots from two of our post-game surveys, showing that the majority of players considered the gameplay to be intuitive.

The below screenshot is the result of our first playtesting session.

![Figure 15: Playtesting 1 Results](image)

As we can see, the majority of the players were able to learn the game’s controls. However, there were some players who struggled. Due to this round of playtesting,
we found that the tutorial level was not sufficient. From our observations, many players were unable to differentiate items from obstacles, while others would rush through to the end of the level without glancing at the tutorial images in the background.

As a result, we made adjustments to the tutorial level based on both our observations and player feedback. Now, the tutorial level contains obstacles (doors) that open only if the player shows mastery of the game’s basic controls.

After making the aforementioned adjustments, we posed the same question to a new batch of playtesters. As we can see, there is a marked improvement in learning the game’s controls. This helped us significantly towards our goal of making the game stand-alone.

![Figure 16: Playtesting 2 results](image)
Related Work

Online math and learning games like coolmathgames.com have the most similar learning goals to M³. Although these websites have a large variety of games to play, our educational game seeks to introduce more advanced topics, such as kinematics. Due to the narrower scope, our game is able to expand in its depth and mechanics without veering into “lesson plan” gameplay.

In terms of gameplay, M³ boils down to a 2D puzzle shooter. A similar game, Bubble Shooter, has the same barebone concept: the player shoots items at a target with the goal of clearing the level. However, M³ is unique to Bubble Shooter in that the player has control on which projectile they use and when they use it. Furthermore, players must be conscious of various physics mechanics, such as frictionless, bouncy, or rough materials, in order to efficiently clear levels.

M³ can also be considered as a light adventure game in later levels. The player is “soft locked” out of certain areas and puzzles if they do not explore a different area prior, as the items they find in said location provides the necessary resources to proceed onwards. This is similar to Legend of Zelda’s temple design, where areas of the temple are accessible only after obtaining the temple’s item. Said item (ie: wind boomerang, iron boots) is key to solving the temple’s puzzles and moving onwards. Likewise, players are rewarded for exploring the level, as doing so will naturally allow players to obtain more fuel crystals and complete the level.

However, M³ differs from most adventure games due to its puzzle-like nature. Players are limited to exploring a singular level at a time, and any items obtained in prior levels do not persist for the next. This can lead to a sense of disconnect for the player's overall sense of progression.

Overall, Seeds in STEM aims to specifically educate Santa Maria middle school and high school students, allowing its curriculum and settings to be tailored to their needs and experiences. Our project also resembles physical STEM games, like Snap Circuits, which encourages children's interest in circuit building. Although our game
is similar in intent and content to such physical games, our project will be free and accessible online, only limited by Internet connection and a working computer.

Future Work

If our team was given more time to work on M³, there are some aspects of the game we would be interested in developing further, as well as some features we would hope to add.

We chose to focus on fewer physics concepts in this iteration of the game in order to refine the mechanics and feel related to the physics concepts. However, given more time to add more complicated levels, we would most likely introduce more advanced physics concepts. These could include spring systems, rotational motion, and inertia.

During the planning and development of this game, the only aspect of accessibility that we focused on was access to the necessary technology. We knew that all of the workshop students had access to Chromebooks, but no access to external hardware (such as a mouse) nor ability to download applications. Therefore, we focused on allowing player interaction through keyboard input only, and ensured that students would be able to play M³ online. Given more time, we would be interested in exploring more possible accessibility concerns. For example, we would consider the following common accessibility issues:

- Visual (e.g., color blindness)
- Motor/mobility (e.g., difficulty using a keyboard)
- Auditory (hearing difficulties)
- Seizures (especially photosensitive epilepsy)
- Learning/cognitive (e.g., dyslexia)²

We would also hold a virtual meeting with the workshop planners to discuss the specific limitations that some attendees might have.
Conclusion

We learned many valuable processes and skills while developing M³. Firstly, we studied and utilized the iterative design process, which is invaluable to game development. Through prototyping, playtesting, evaluation, and refinement we were able to center the game around the players and indirectly include them in the decision-making process.

Further, we worked with two teams to carry out this project. We partnered with the LAES 411 (Technology and Community Engagement) class to adjust the game’s concepts, length, gameplay, and story to the needs of their workshop. This involved presenting to the 411 class multiple times and discussing our ideas for the game. We were also lucky enough to work with three other student game developers in our CSC 371 (Game Design) class. They helped tremendously with the iterative design process, as well as with the large amount of technical work that went into M³. Through working with these two teams, we improved our project management, communication, leadership, and decision-making skills.

Overall, we are satisfied with our final deployment of M³. The game is in a stable state and incorporates several physics concepts in a subtle and entertaining way. However, we would like to have put more effort into the iterative design process. For example, we could have playtested with more users and more representative users (high school-age students). We would also have liked to playtest more often. Working virtually has definitely limited our interaction with playtesters and workshop attendees, and our ability to meet with teammates in a more collaborative environment. Altogether, we completed our goal for this project: to create an educational game that prioritizes fun and meaningful play, and encourages players to take an interest in STEM.
References