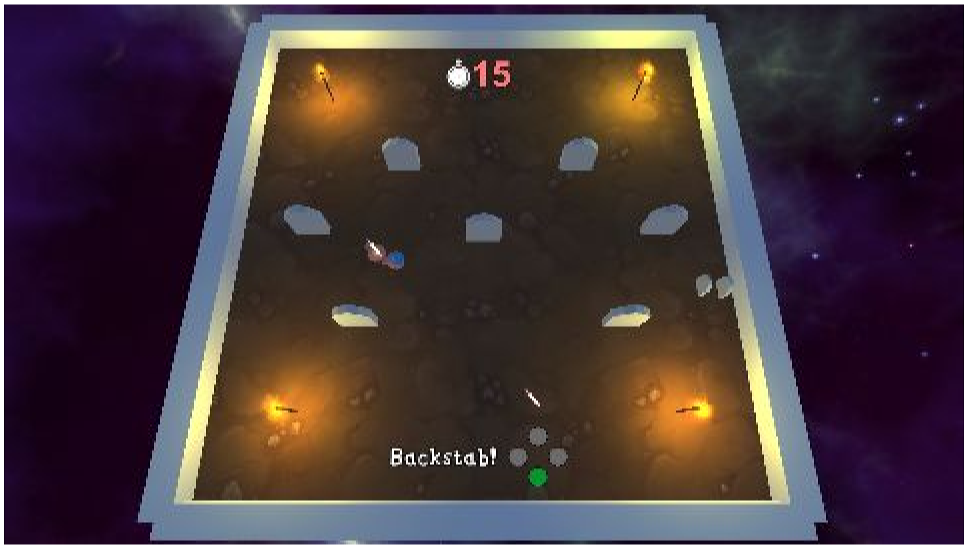
# Grudge Trudgers Report

By Stan Huang, Alex Zhu, and Didi Aburaad

## Overview

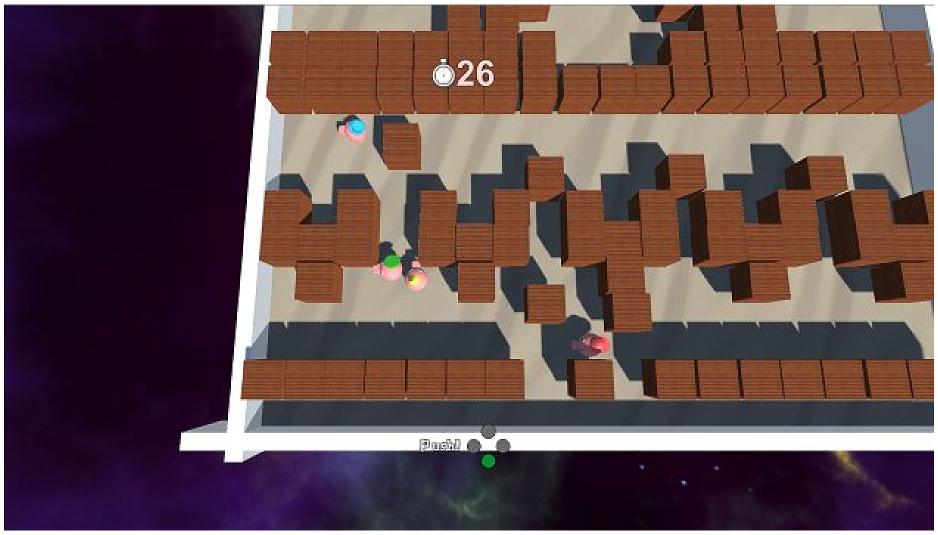
### *Description*

Grudge Trudgers is a top down 2.5D four-player party game that takes advantage of human beings' ability to switch between cooperating and competing with each other in an instant to create an experience where the four players will have to navigate through a variety of different stages/minigames while attempting to individually collect the most gold possible. Among a total of 15, Grudge Trudgers consists of four types of minigames with the stipulation that every game must give players the flexibility to either cooperate or compete:



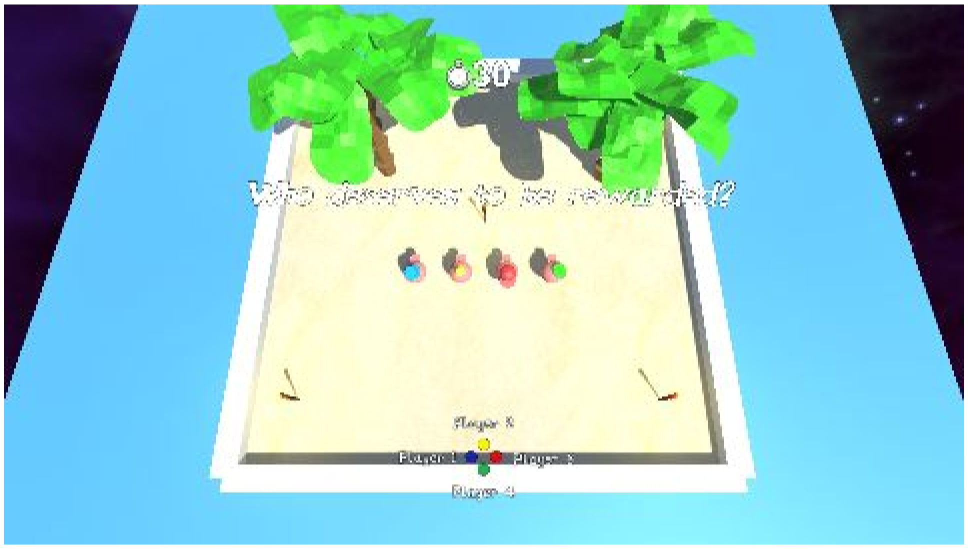
*Players can backstab each other for coins in this level*

1. **Action -** these games require the players to move around and interact with their environment and each other in order to complete some goal in a given time frame. These goals become easier if players cooperate. Action games can end up with either winners or losers. Losers are punished while winners are rewarded.



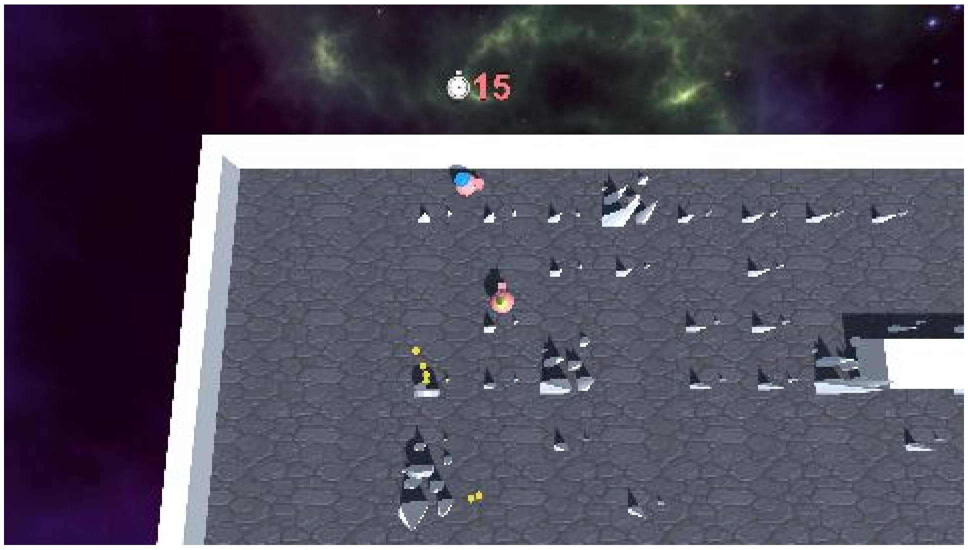
*Player push boxes together to solve this puzzle*

2. **Puzzle** - These games require players to solve some sort of puzzle in order to advance to the next stage. Players can either cooperate with each other in order to more easily solve a puzzle, or do it individually and attempt to leave other players as losers.



*Players choose one player to be rewarded*

3. **Choice** - These simple games only require each player to make a choice from a possible four choices (as denoted by buttons). The results of the four choices can affect the consequences of the game, such that some players might be punished and some players might be rewarded. Cooperation will usually lead to better consequences for all involved, but players can easily target others. Active discussion between players is encouraged.



*Players avoid spikes and try reaching a goal*

4. **Obstacle -** These games require players to get from one end of the level to another end designated by some goal. Players must get to the end by a certain time limit, and players can either assist or interfere with each other.

## *Narrative:*

Within Grudge Trudgers,  players are treasure hunting pigs trying to escape a dungeon filled with perilous traps. Being the greedy pigs they are, players are encouraged to both work together or sabotage their fellow players in order to earn gold to win the game. Each minigame gives our players the opportunity to either earn or lose gold, and player gold count is hidden from everyone until the game is completed.

Each minigame within Grudge Trudgers follows a different theme and offers its own unique experience playable with limited instructions and simple controls.  Levels in the game were either set up to be puzzle, action, or choice focused.  Action games in our design include a level where players paint the ground where the player who has painted the least loses gold, a level where players have to cross a bridge without being blown off by wind-tunnels, and a level where spikes randomly pop out from the ground preventing players from reaching an end goal.  Puzzle games in our design include a level where you have to navigate a dark maze and staying together illuminates yourselves.  Choice games are styled to have each player choose with the buttons certain players on their controllers depending on the level.  For example, one level is set up that players can choose which player to punish, i.e. each player is allowed to pick yellow, red, green, or blue are dictated by the xbox controller scheme.  Whichever player garners the most votes will lose gold.

## *Inspiration:*

From the very beginning, we as a group wanted to employ a social aspect to our game. Being avid fans of cooperative and competitive multiplayer games, we hoped to create an experience that would allow multiple players to have to interact with each other. However, we could not decide on whether we wanted players to compete or cooperate with each other, as either would make for a fun experience. In the end we wondered, why not both?

We drew a lot of our inspiration from party games such as *Mario Party*, *Mario Kart*, *Overcooked*, and even *Monopoly,* all games that we as a team loved to play. We really liked the frantic cooperative aspects of *Overcooked*, and the mini-game style *Mario Party* offers. *Monopoly* and *Mario Kart* were not designed specifically with the idea of players intentionally sabotaging one another, but gradually the culture evolved where sabotaging and forming grudges with friends and family (and friendships) have become the norm.

What Grudge Trudgers does is try and turn those player grudges, alliances, and betrayal, into a core aspect of the game.  While only a cultural evolution in games like *Mario Party* and *Monopoly*, We wanted players to know that it's not necessary to perform their best in order to win, but rather they should make sure that other players don't perform any better.  Working together is fine and dandy, but betraying and ruining your friend's lives is even better. Nobody can win if everybody loses in Grudge Trudgers.

### *Experience:*

Overall, we want players to have fun while learning a few new things about each other. WIth the social aspect of the game being the most important and apparent feature, players should be constantly speaking to one another, whether that be discussing stratagems or yelling curse words at each other. Any sort of player interaction is good interaction, and will help the game itself feel more alive as it becomes more than just a virtual experience, but rather an extension to the interactions between the players themselves in real life.

As for the game, players should be wary and cynical of each other. They should be questioning who to trust, and be constantly keeping track in their heads who might be in the lead, and who has caused them the most harm. They should be reaching for every last coin and pushing each other out of the way to reach certain goals. They should be forming alliances and betraying each other when the time is right. In this sense, Grudge Trudgers really is a game about harboring grudges, and hopefully players themselves are not shy in playing up those emotions.

# Design

After coming up with the concept of leading the players towards loving...hating...our game by pitting friends against each other, we came up with the name "Grudge Trudgers." We pictured dungeon-diving, armor-clad pigs clamoring to gather as many gold pieces as their greedy hearts desired, resulting in grudges forming between the participating parties. With this in mind, we tried to create a character matching our game's envisioned aesthetic. Although the armor was eventually swapped for hats, the original pig figure stayed true.

## The Roundest Boy, Part 1: Brainstorming

While sketching out the pig (fondly referred to as "The Roundest Boy, or TRB for short), Didi looked for feedback from the rest of the team. They were concerned about the size of his head, since the game would be top-down and the rest of the body would not be visible. But what better way to identify your player than as a bobbing pink orb with a stout snout and snazzy hat? After thinking about it some more, we decided to keep his original top-heavy form. This later proved handy when we had trouble getting his animations working, since it was difficult to see his scurrying feet and swaying arms anyway due to the top-down view in the game (later we found that the animations had exported under different names and got them working).

## The Roundest Boy, Part 2: Conception

TRB came into this cruel world over the course of several weeks via the use of Blender. After a couple rounds of sketches, TRB was made in Blender as a three dimensional model, then given texture, rigged, and finally animated. TRB was made with only two animations: idling and running. Since our characters were relatively small on-screen and didn't interact too much with the world, we found these to be the only animations we needed for the levels we designed.

## Character/Color Continuity:

We designed our entire character experience to be easy to follow by aligning the colors of the controller buttons with the colors of the players' hats; red, green, blue, and yellow. This consistency was furthered by giving visuals for players to follow level by level; anytime control buttons needed to be used in a level, we provided a visual at the bottom of the screen saying what each button did for that level.

## Strategic World Construction:

The actual playable space of our game was simple and straightforward. Playing off our original intent to make this a dungeon-diving game, each level was meant to present itself as a new 4-walled room to complete a task in in order to continue. Again, we wanted our game to be consistent throughout, so we kept the rooms' basic layouts as similar as possible. We designed the setting and obstacles within levels in order to encourage as much interaction between players as possible. For example, the Box level was made so that a 1-box open space can only accommodate one player, causing frustration for players that get stuck behind an especially slow player. We also tried to make it clear what the easiest path would be in order to motivate all the players to try to traverse it simultaneously, only to have them find that crowding the space is not always conducive to a win.

## Music:

The music for each level was chosen by Alex and was strategically picked to give the players a feel of how each level would play out. For example, levels that would be a stressful race or puzzle played upbeat, high tempo music to instill a sense of urgency in our players. Levels where players make decisions had slower, more placid musical themes in order to relax players and make sure their decisions were well thought out. A good thing to note here is the need to make sure audio tracks are thematically similar to solidify the game's identity as a frantic, light-hearted party game. I.E. most of the tracks used in the game are mostly acoustic-based with emphasis on high register melodies with bright-toned instruments.

### Sound Effects:

Our sound effects were chosen to make it clear that they mimicked the actions of the world, while maintaining a playful feel. Rather than a stab action in the Backstab level sounding like a knife entering flesh followed by an anguished pig squeal, we decided to stick with a simple knife draw sound. We did not include any pained sounds whatsoever -- we wanted to keep the game upbeat and fun, if not destroying friendships in the process! To further our positive vibe, we included an oink button for when players felt especially “in-character” or wanted to express themselves.

# Implementation

The implementation of Grudge Trudgers made use of two vital tools:

1. Unity - This classic beginner’s tool to game development was used in conjunction with a slew of C# scripts to handle all the logic of the game.

2. Blender and Gimp - These open source art tools allowed us to model, texture, and even animate our own characters and objects that we used in our game.

## Unity

With our game concept utilizing 15 separate mini-games, we needed to create a consistent layout for our games that could be reused. Before even starting development, we agreed upon a level design guide that each minigame must follow. This guide specified 4 basic objects that each minigame must have:

1. **Players** - The minigame must have our 4 players somewhere in the scene with the correct tag.

2. **LevelLogic** - The minigame must have an empty GameObject containing the level’s “LevelLogic” script in order to handle each minigame’s level logic. This script was also responsible for determining player punishments and rewards, as well as setting up the level itself.
3. **UI Canvas** - The minigame must have a UI Canvas element that contains all the relevant UI elements it might need. This includes timers, instructions, images, and button layouts. The minigame’s LevelLogic script is also able to access the UI Canvas elements and change them when needed.

4. **Camera** - The camera is responsible for viewing the actual game scene and is either fixed or tracking. If fixed, the camera sits at a 70 degree angle and views the entire game scene without moving. If tracking, the camera keeps all players in the field of view and moves to center itself within the bounding box of the 4 players.

In addition to these 4 consistent elements, we needed to write various scripts, tune our lighting, and utilize audio sources.

***Scripts***

Scripts made up the majority of our development and were primarily written in the C# language. Within the scripts we had behaviours which controlled how certain objects reacted such as pushable boxes and goals, and controllers which controlled how different objects moved such as wind turbines and the players themselves. With the player controllers, we found that we had to link them up to an XInput library in order to get player movement mapping to an xbox 360 controller to work.

With the level logics, we utilized states that specified where the level currently was at. During the opening, the level logic state would be set to “begin”, same with “gameplay” during gameplay and “punish” during the punishment scenes. This allowed us to use a consistent level logic format between every minigame and know where to commit certain actions such as starting the timers and loading new levels.

***Lighting and Shaders***

Lighting varied between almost every minigame. We found that since we used different thematic environments for each level, we had to vary the lighting to match. Some levels required the use of bright almost universal lighting to allow players to see everything. The box level is one such example of this, as we wanted players to see the entire level ahead of them. This required us to utilize bright almost toon like shaders that lit up textures. On the other hand, in darker levels such as our maze level, we needed to use lighting that was not ambient at all in order to get it pitch black, and attach point lights to the players themselves. Each level had its own unique lighting, and had to be implemented in its own unique way.

## Blender

1. **Modelling** - Adding shapes, then editing these shapes by extruding faces, adding faces/edges, deleting vertices/edges/faces, rotating, scaling, subdividing surfaces, adding modifiers, and otherwise manipulating the shapes on screen to best resemble the model at hand.

2. **Marking Seams/Unwrapping** - Highlighting edges as if they were lines to cut your model along, in order for the model to lie flat for more even coloring/texturing (without warping). The final result after unwrapping is called the UV layout.

3. **Coloring/Texture** - Taking the UV layout, color it using Gimp and then save it as a .png. Use this .png in Unity to add the texture to the model.

4. **Rigging** - Adding a root bone, extruding that bone to create hips/shoulders, extruding those bones to create arms/legs. After completing half of the armature, mirror it back to the other side and make sure the entire structure is connected where it’s supposed to be.

5. **Animating -** Marking keyframes at the major points of the desired animation, then letting the animation play through to see if smaller changes need to be made between keyframes.

6. **Exporting -** Export the entire model, animations and all, as an .fbx file to use in Unity.

### Who did what?

**Didi** acted as the head art designer, creating the character and additional necessary models. She also found sound effects and ensured consistency for the sounds applied to in game actions. Lastly, she contributed to level design, creating the layout for 2 levels.

**Stan** acted as the game's lead developer and handled most of the design and implementation of the game's scripts and scenes. He also drew up and implemented the main technical architecture of each minigame and script, and was in charge of enforcing a certain consistency between different aspects of the game. As the only member of the team with prior experience in game design, Stan also assisted the others with learning the ins and outs of both Unity and Blender as well as getting them started with a variety of resources.

**Alex** was the game's secondary developer, contributed to level design, and was music designer for the game's soundtrack. He contributed to level design and scripting for at least 2 levels, and got multiple controller support to work within Unity's finicky input manager. Alex was lead on soundtrack to enforce a consistent theme throughout all the levels.

## What Went Right?

Didi's spiel:
Blender:
A big part of the process was me learning to use Blender. Over time, I got pretty fast at turning out models, so long as they were stationary and simple to unwrap as a flat surface for future coloring/texturing. What took the longest amount of time was first getting used to Blender. Although Stan directed me towards an extremely useful and relevant tutorial, it was difficult to remember which shortcut/tool did what. The tutorial was very specific about what each tool in Blender did, but it was also rather dense to sort through, as Blender has a boatload of features that must be used in a certain order to prevent problems with rendering later on. I had to go through this tutorial, a quest in its own right, around 4 times before I became familiar with the bare basics. Even so, I often found myself going back to it as a reference for when I forgot a specific tool. However, this tutorial set me up to efficiently make models throughout the quarter. It taught me Blender terminology, later allowing me to accurately search for advice online about tools outside the scope of the original tutorial. Given the complexity of Blender, it would have been extremely difficult to get started if it weren't for this first tutorial showing me the ropes. I really feel that without the foundation it provided, I would've spent hours searching for the basics without getting as complete of a picture as I did. Although it was not all-encompassing, it truly prepared me to be able to expand my skills and learn more about how to use Blender.
Gimp:

The tutorial also reintroduced me to Gimp, a tool I had downloaded a long, long time ago and then proceeded to forget about. Although not super in depth, the tutorial showed me the bare basics very quickly. Many of the tools were rather intuitive, unlike Blender, and so the learning process was much faster here.
The Dynamic Duo:
With these two tools, I made model after model of TRB. Some of the largest difficulties I encountered were joining multiple shapes within one model, merging vertices, making a set of vertices round, marking seams/unwrapping, rigging, and properly naming animations. I was only able to overcome such difficulties because of the foundation I already had in place.

In General:

Our team met on a regular basis for several hours every week to check in and make sure we were all up to speed. These meetings kept us focused and motivated us to try and complete things faster and better than before, while also allowing us to express our frustrations and find solutions together.

## What Went Wrong?

### Unity

One big bug we found but couldn't fix in time for the demonstration was an issue with Unity's particle systems. To create the wind tunnels in the bridge level, we assigned particle systems and large invisible colliders that would be active triggers when the particle system is active. So when the particle system fires out wind particles, players should be pushed in the direction of the particle system. The bug we encountered was the trigger would persist for a second or two after the particle system would be turned off; after further retrospection we realized it was an issue with Unity's particle systems and how it updates its "isPlaying()" and "isStopped()" boolean tags. Unity will update such boolean tags when the particle system is stopped when all particles disappear from the system, not when the emitter stops when writing it to a scripted routine. In order to fix this bug in this game and future Unity projects, setting particle lifetimes to a lower value will resolve this issue as this means all particles from a particle system will disappear faster after the emitter is turned off.

### Blender

Within Blender, a problem that consistently seemed to appear was when we would start by adding a circle and then extruding it upwards along the z axis. This action rendered the model partially transparent in Unity. Given the simplicity of the models we were working with, it was easier to redo the models by starting with a cylinder rather than go through the mess that is problem solving within Blender. After a bit of digging, we eventually realized that the problem lies in the normals of the model -- by enabling backface culling and then recalculating the normals, we could have reached the same solution.
Another problem encountered in Blender was finding the animations that we created. After exporting to .fbx and adding TRB in Unity, the run animation would not work. The tutorial stated

that when adding a 2nd, 3rd, *x*th animation, that a certain button must be pressed to make sure that *all* of the animations get exported with the .fbx file. Thinking that this was the issue, we went back and tried re-exporting after pressing the button, with no success. We were about ready to go back and recreate the model entirely from scratch when we realized that there was an extra animation that exported with our model. This animation, with the generic name “Action,” contained the running animation that we were looking for!

### Generic

Another issue, to be expected, was meeting the visions of all the team members. For example, agreeing on TRB’s head and body proportions, deciding on the layout of obstacles within levels, and of course, picking the name of the game itself (it’s not a pretty name, and apparently it is unnecessarily difficult to pronounce). Luckily, we are all very agreeable, nonaggressive people who can negotiate to come to a consensus.

## Ideas For Next Time

During our demo, we found that many players were confused by the instructions provided at the beginning of each level. What we thought were clear instructions on how to play were actually not so clear. Going forward, we would do more play tests to check the user’s understanding of how to play each level.

On this note, some of the players did not seem to have finished reading the instructions by the time the level actually began. Rather than throwing the players straight into the level, it would be ideal if we got the perfect balance between our players understanding what a level entails and having the freedom to play the level as they chose.

From the very beginning of the game, it was not made clear enough to players which number/color they were. In the future, we would like to further standardize how we refer to players (cutting out player number entirely and relying entirely on color, for instance) and allow them to interact as their character before beginning the game.

Players wanted to replay or choose specific levels to play. We could easily implement this going forward by simply changing the first menu to allow users to choose levels to play.

One thing we would like to focus on more next time, but we didn’t have the time to devote and polish was the UI. When some people played the game, they were confused as to what a mini-game’s win condition was, or which buttons did what. When developing a new game, we’d invest more time into creating a tutorial, and making button prompts more prominent when displaying instructional text at the start of each level, or when contextually relevant. Additionally we would implement a sort of counter to allow players to keep track of many mini-games they’ve already played.

Another issue in some of our levels and how our camera was coded was that when players would get too far away from each other the camera would be zoomed very far out in order to keep everyone in view. Examples of this would be in the maze, box, and soccer levels where the issue was very apparent.  In next iterations of the game we would look at implementing split-screen in these levels to lessen the strain and awkwardness caused by our current camera.