Interim Demo

In this document we want to describe our reasoning and progress during the development and showcase what we achieved.

Roller Coaster Movement

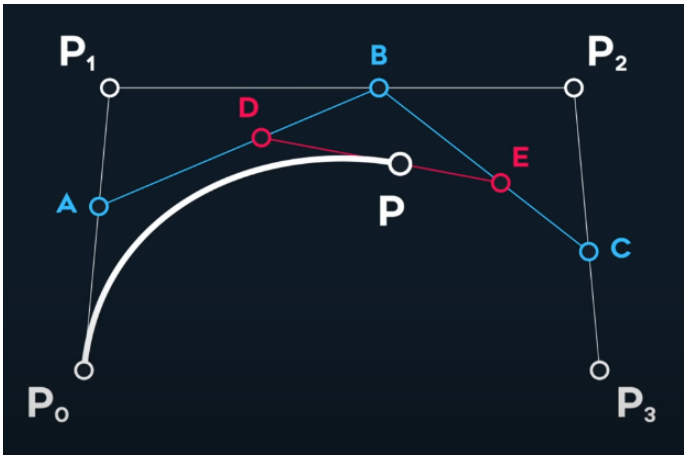
One of the main features of the game is the chaotic movement of the train. We wanted it to be as extreme as possible, while maintaining a feeling of control for the player and without breaking the AI trained enemies completely. Finding the right balance between fun and frustration will be a challenge, but we hope to get some good feedback in playtesting in order to tweak it to perfection.

Implementation

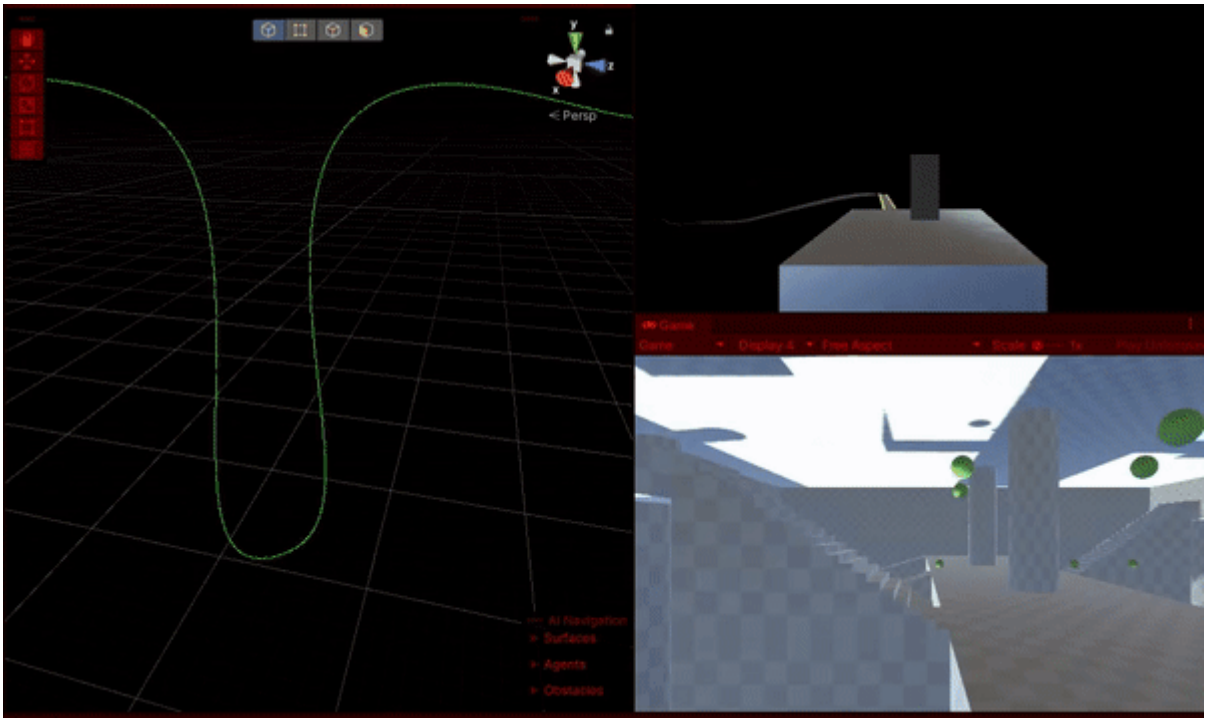
For implementation we had two different ideas we wanted to explore. The first one was to move the roller coaster like a real roller coaster and see how gravity and rigid body physics affects the objects inside the roller coaster block. The hope was that it already works pretty well and we can use unity physics to our advantage. The second option was to have a static level that doesn't move at all, but at fixed times we apply a strong force to all objects inside the level to mimic an extreme movement. Since we had a feeling that the first option had the potential to fail, we kept this option in our backhand, in case something went wrong, but first focused on testing and implementing the first option. In a later stage we also thought about combining the two options in case the real movement has too little effect on the movable objects inside the roller coaster.

Real Roller Coaster Movement

There are multiple advantages of implementing the roller coaster movement as close to reality as possible. First of all it is easier to design a level with visual feedback of the tracks for the movement. Furthermore we can use the movement as visual cues to tell the player we are moving and implement immersion way easier by simply adding a nice background and implementing windows. And lastly we hope by doing so the physics affecting the player and enemies stay as close to reality as possible, too. But how did we implement it? For now we used splines aka cubic bezier curves to model the roller coaster track. A cubic bezier curve in its simplest form is a set of nested linear interpolations between 4 points in space. Picture 1:



In Picture 1 for example you see a basic example of a cubic bezier curve. First you interpolate on the three lines between P0, P1, P2 and P3. A interpolates between P0 and P1, B between P1 and P2 and C between P2 and P3. Then you connect those three points and interpolate between those. Here for example D then interpolates between A and B and E interpolates between B and C. Finally you add a Point interpolating between those last two point D and E, which gives you P. P then can describe any curve depending on the positions of P0-P3. Now adding tangents and normals to the points gives you a complete spline with roation/orientation, so you don't just make objects follow a path but also adjust their rotation according to the spline. Luckily we didn't have to implement all of this logic ourselves. That alone would have taken a big portion of our development time. We found a Path Creator tool (credits to Sebatsiona Lague and his project) that we used to create our roller coaster track. We had to modify it a bit to our needs and had some bugs we fixed, but basically we had a finished tool to help us create the tracks. So now we needed to model them and add a Level to it that would follow the path and the orientation of it. For that we first used a simple cube with some balls inside to mimic movable objects/enemies and let it follow the path. We then tested how the curvatures impacted the ball movement and then created multiple prototypes of curves we could use as level. In order to have a basic working level that can be used for training of the enemies we made one with fairly simple movement, to make the training easier. In the future we want to go crazy with it, but we need to find a way to efficiently train the enemies in such an environment without the training taking multiple days. But more on that in the ML-Agents section. Her an example of an intermediate development step:



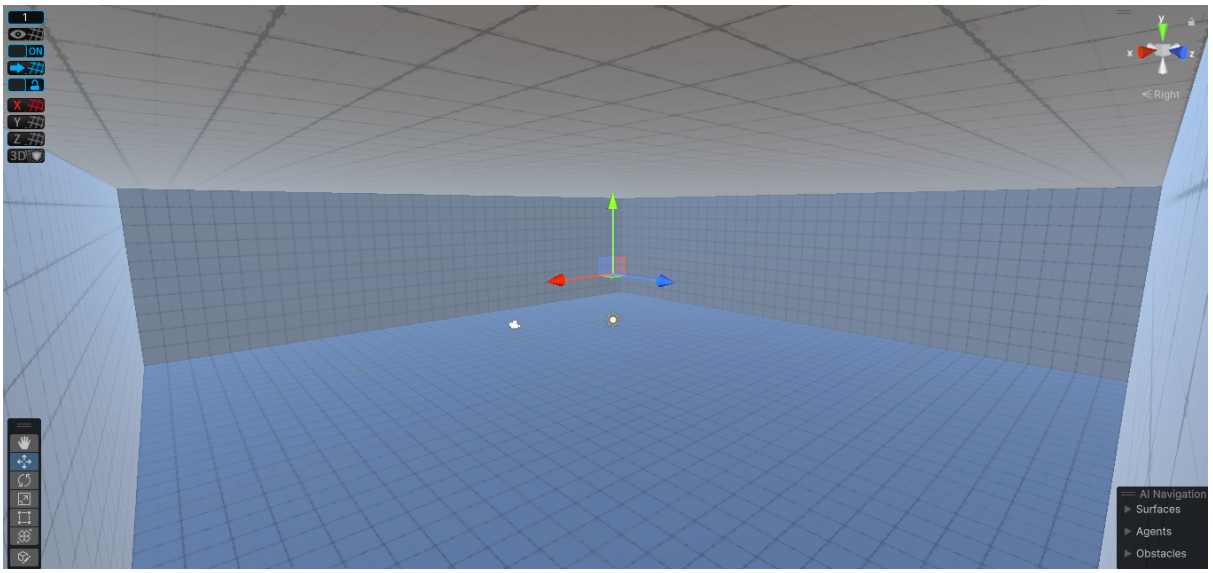
Basic Level Design

To make designing the levels fast and easy, we opted out to use mainly the ProBuilder package of Unity. Additionally, to make the manipulation of objects created with Probuilder easier, we used ProGrids. This allowed us to precisely control the placements of the objects, their heights, widths and lengths. When, for example, an object such as a cylinder is being resized, ProGrids allows the user to increase the height by 1 unit precisely in the chosen

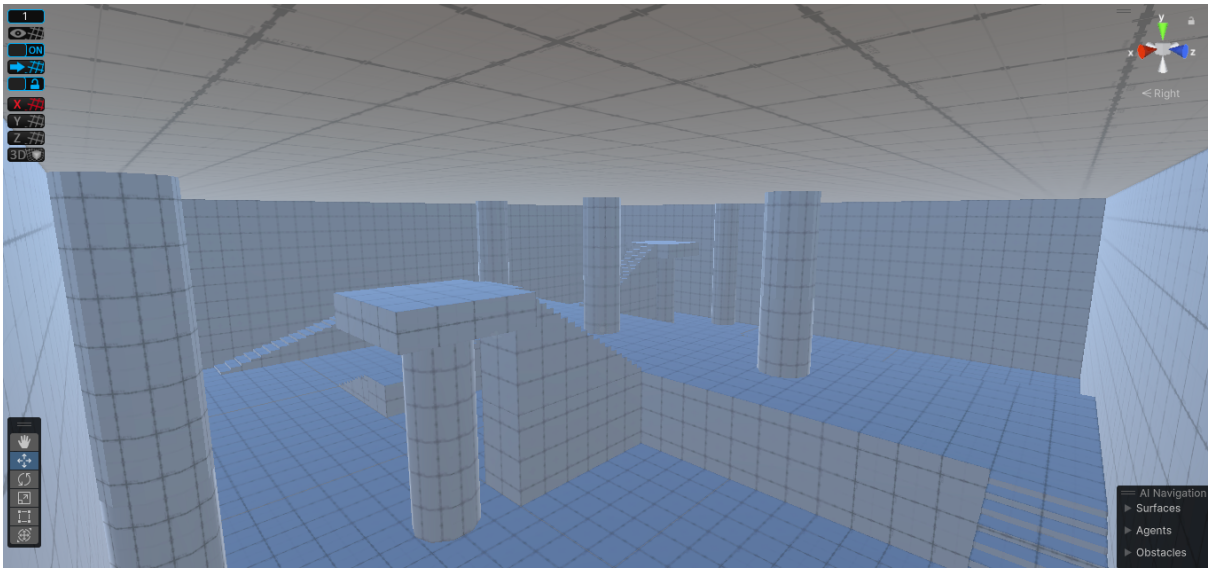
axis. This is especially important for our project as the ml-agents package and machine learning in general requires very precise and controlled environments for successful training (no accidental small gaps, same orientation etc.). PolyBrush is used to manipulate objects and textures in a way similar to painting which again makes prototyping easier and faster. One of the most useful tools of PolyBrush is UV painting which allows the user to paint different textures to objects.

Using these tools, we build a complex and a simple level to train our ml-agents in. These are as follows.

Simple level:



Complex level:



The challenges of working with ProBuilder is that while it is good for rapid prototyping, it is difficult to make precise and detailed levels with it. Also, it has a steep learning curve as things get more complicated and erratic if you try to push its limit. It took 10 - 15 hours to be able to learn it well enough to properly use it. Due to its limited capabilities, it requires a lot of

hacks to get certain things right and it is difficult to learn these tricks since documentation is very lackluster. The challenges of working with PolyBrush are again very similar and the challenges of working with ProGrids mainly come from the fact that it is no longer officially supported and left in a limbo even though it is required to create precise levels.

Regarding the initial plan, we finished the basic level design and are currently improving it by detailing it out. We are currently testing moveable objects with rigidbody physics to see how they fit in our gameplay and how to implement them the best way. Due to the difficulties of using ProBuilder while creating precise and complex objects and structures, we decided to use Blender for that purpose in order to create ornaments such as a chandelier etc. We plan to use ProBuilder only for rough level design and then export it to Blender to detail it out.

Player Movement

Implementation

We decided early on that a character controller based on forces and impulses would be preferable based on the nature of the game. This way the player can interact in a more natural way with the highly chaotic environment. Currently a simple controller which allows for moving, jumping, controlling the camera and shooting is implemented. It is currently in a functional state, but the exact values will require more fine-tuning as we go on in the development of the game. We think that the ability of the player to control the character well and precisely will be essential to them being able to navigate the chaotic environment without becoming highly frustrated.

Additionally, the current implementation only works for keyboard and mouse, but since it uses Unity's Input system, it becomes fairly simple to implement functionality for gamepads as well.

Challenges

One big challenge became immediately apparent when trying to merge the controller and the moving map, it became extremely difficult to control with the camera and player character not moving as expected. It turned out that character controllers usually work with the assumption that the world is static and that the down direction remains the same throughout the game, which is definitely not the case for our game. On top of that, we became aware of this issue later in the milestone as the controller and the map were developed separately and then brought together, leaving us little time to address this issue.

While not perfect right now, we were able to solve the issue in a relatively good way. Now the camera aligns with the level and its current rotation, making it much easier to see and keep track of what is going on in the level. The movement also aligns with the level, so the player and camera can be controlled well. In case we want to implement full rotations of the map this would need to be modified somewhat, but we do not anticipate this to be a very big issue.

Weapons and Gunplay

Our main goal is that each weapon the player can use feels unique and can be used either to disrupt the enemies or move around the level with ease, preferably both. Currently we have 2 functional weapons implemented into the game, with tools in place to easily create more from a technical perspective. There are no graphics implemented yet, only placeholder shapes which will be changed later.

First is our main weapon, which launches a projectile at high speed. This can deal damage directly to enemies but also will displace them by applying a force upon impact. It also applies a force to the player, pushing them in the opposite direction, which can be used for retreats or super jumps. One defining feature of this weapon is that it can be charged by holding the button. It can gain up to 2 levels of charge depending on how long it was held, each increasing the size, damage and forces of the projectile.

Our second weapon fires a burst of 5 small projectiles that are not affected by gravity. They move a bit slower and do not deal damage on impact. However, when they hit an object or enemy they will stick to it and, after a small delay, they will detonate, dealing damage and pushing the object hit away. This weapon has a delay before having an effect and is currently somewhat imprecise with shooting, but it can deal high amounts of damage if all projectiles hit an enemy.

AI-Part

As a reminder, our main goal is to design a physically based "mannequin" or virtual robot for the enemies that responds properly to changes in the map (map rotation, acceleration, etc.).

We modeled two different enemy characters from scratch in Blender to have full control over limb splitting (for later use in limb separation) and body anatomy. Due to a lack of experience in building a flexible pipeline for iterating the character, we decided to make the 3D model as good as possible from the start (as opposed to a simpler model, but that would require a similar amount of refinement in Unity anyway).

The modeling process was very tedious (again, we lacked experience with Blender).

The problem was the use of Blender itself and the lack of know-how about good modeling techniques: so simple tasks became time consuming:

- How to apply a change to all selected vertices?

- Forgot to apply "Apply Transform" before adding to Unity, resulting in unexpected behavior (inconsistent scaling + rotation creates a shear matrix for the children that deforms them in strange ways).

- How to foreground reference art and make it opaque (amazingly, this took us about 30 minutes).

Basically, every step had to be looked up, but using simple primitives was not an option due to our intended art style.

We managed to create a character that fit our intended concept art. Feedback from our playtesters was also positive about the aesthetics of our model.

At 250,000 triangles, the model is considered high-poly and would need some retopologizing to reduce the number of vertices (perhaps combined with normal backing to preserve detail).

The Enemy Spider has a DoF of 48, so it is quite flexible.

A second Enemy, simpler in terms of DoF and polycount, was introduced to simplify prototyping (16 DoF), using the same script for logical handling.

The first step, the "Hello World of Virtual Robots" so to speak, is to test the rig to see if it can move to a certain position on a flat plane by applying torque to its joints. For this, we use and optimize Unity's existing script.

The virtual robot receives as input the position, the local rotation of the joint (relative to its parent concatenated joint), and several rays searching the ground. The configurable Unity joints use a PD controller to achieve the desired rotation. In our tests, the custom PID controller provided no advantage because the motion is very fast (so the integral part can't build up anyway).

For our first opponent, we used an insect-like rag doll with four legs and 2 joints each, moving on a flat plane. The reward is to move at a certain speed (between 0.1-20 m/s) and keep your head gaze on the target. The two reward functions (which are both between 0 and 1) are multiplied to force optimization of both functions.

One study has shown (https://arxiv.org/abs/1812.07035) that in motion matching applications, 6 parameters instead of quaternions for the rotation representation lead to better results; we will further test if this will help our case.

Choosing the right reward function

Reinforcement learning finds very creative ways to optimize the reward function or in other words to “cheat”, making its definition a very tedious task: After each iteration of the code, it takes about 3 hours on our fastest computer to see a result.

Best slips:

- If you set the penalty for touching the ground too low, the spider will turn on its back and run this way because it is more stable with this running technique while keeping its eyes on the target.

- If you set the ground penalty too high, the spider will constantly jump off the cliff/plane because the drop penalty is too low.

- If the wrong forward vector was accidentally set, it learned to walk backwards.

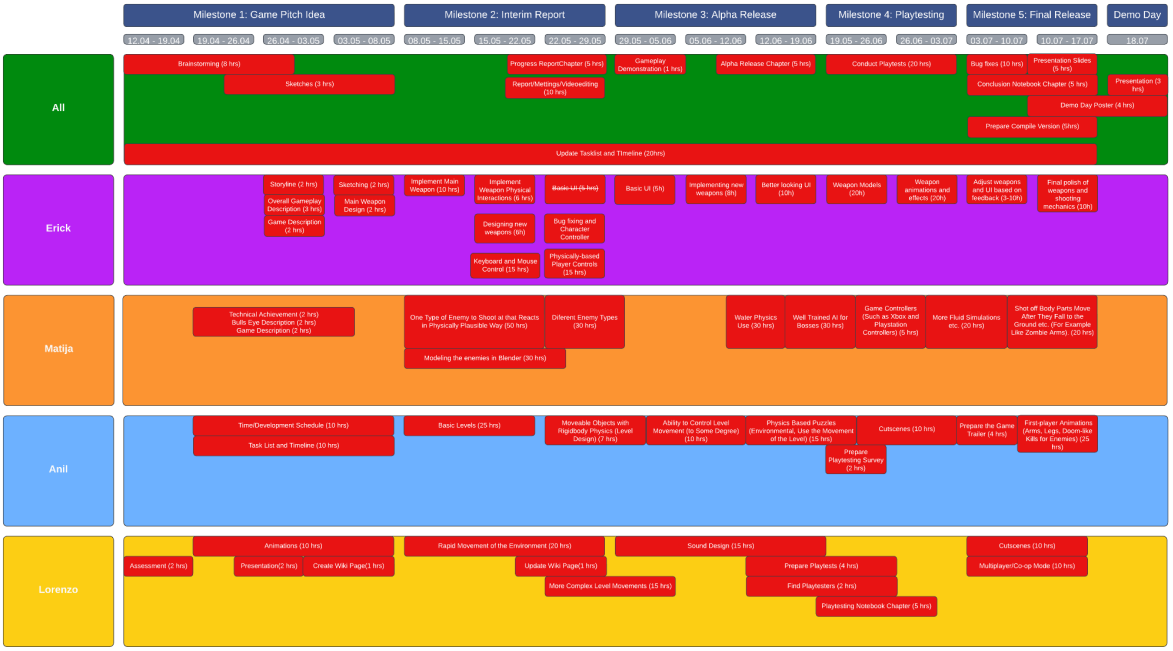
- When the reward was the reverse of the distance to the target (1 for very close), the spider moved as close as possible to the target but did not touch it to maximize points. Setting the reward for reaching the target to “Maxsteps - CurrentStep” solved this problem.

Summary

We have met our goals in the timeline, but only with a severe increase in work time due to unpredictable bugs.
We have not managed to integrate all parts together in one map (also bc our Git repository got to big the day of the deadline when we wanted to integrate every part), but are very close to do so.

Our next steps will be finalizing one map to work and try to make even more complex Enemy types, weapons, and maps.

Updated Timeline



Updated Task List

Tasks	Developer	Time
Brainstorming	All	10
Game Description	Erick	6
Storyline	Erick	2
Technical Achievement	Matija	4
Bullseye Description	Matija	6
Overall Gameplay Description	Anil	2
Time/Development Schedule	Anil	10

Task List and Timeline	Anil	4
Assessment	Lorenzo	2
Updating the Wiki Page	Lorenzo	2
Presentation Slides	Lorenzo	2
One Type of Enemy to Shoot at that Reacts in Physically Plausible Way	Matija	20
Basic Gun Play	Erick	10
Basic Levels	Anil	25
Rapid Movement of the Environment (Only a Few Like e.g. Rotation)	Lorenzo	15
Moveable Objects with Rigidbody Physics (Level Design)	Anil	7
Keyboard and Mouse Control	Matija	15
More Complex Level Movements (G-force, Free fall, Sudden Turns, Acceleration)	Lorenzo	10
Projectile Weapons (Push Enemies, Break Things, Cannon Balls, Spikes etc.)	Erick	20
Physically-based Player Controls (Impulses, Force, Momentum)	Matija	15
Ability to Control Level Movement (to Some Degree)	Anil	10
Mutual Critiques	All	1
Determining the Art Style	All	10
Determining the Gameplay Demonstration Content	All	2
Prototype Preparation	All	4
Feedback Consideration	All	1
Iterating over the Game Idea	All	2
Prototype Notebook Chapter	All	8
Updating Time/Development Schedule	All	4
Updating the Wiki Page	Lorenzo	1
Gameplay Demonstration	All	1

Presentation Slides	All	1
Conduct Playtests	All	20
Prepare Playtests	Lorenzo	4
Sound Design (Movement of the Roller Coaster, Music, Weapon Sounds etc.)	Lorenzo	20
Weapons Have Accurate Knockbacks and Recoils (Such as Rocket Jumps)	Erick	15
Different Enemy Types	Matija	30
Water Physics	Matija	30
Breakable Objects (Glass, Boxes, Destructible Environment)	Lorenzo	10
Progress Report Chapter	All	5
Updating Time/Development Schedule	All	
Updating the Task List and Timeline	All	20
Updating the Wiki Page	All	6
Gameplay Demonstration	All	10
Presentation Slides	All	5
Story and Characters	Anil	10
Physics Based Puzzles (Environmental, Use the Movement of the Level)	Anil	15
More Unique Weapons (Gravity Gun, Saw Blade, Bombs, etc)	Erick	20
Nice Art and Assets	Erick	10
Well Trained AI for Bosses	Matija	30
Game Controllers (Such as Xbox and Playstation Controllers)	Matija	5
Alpha Release Chapter	All	5
Updating the Task List and Timeline	All	20
Updating the Wiki Page	All	
Gameplay Demonstration	All	1

Cutscenes	Anil	10
Multiplayer / Co-op Modes	Lorenzo	10
First-player Animations (Arms, Legs, Doom-like Kills for Enemies)	Anil	25
More Fluid Simulations etc.	Matija	20
Shot off Body Parts Move After They Fall to the Ground etc. (For Example Like Zombie Arms).	Matija	20
Weapons that Use Fluid Dynamics Like a Water Jet Gun etc.	Erick	20
Weapon Progression Such as Adding Mobility Skills to Weapons (Crossbow Provides a Hook etc.)	Erick	40
Playtesting Notebook Chapter	Lorenzo	5
Email to the Supervisors the Contributions	All	
Prepare Playtesting Survey	Anil	2
Find Playtesters (Minimum of Five Participants for Playtesting)	Lorenzo	2
Updating the Wiki Page	All	
Presentation Slides	All	
Conclusion Notebook Chapter	All	10
Prepare the Demo Day Poster	All	4
Prepare the Compiled Version of the Game	All	5
Prepare the Game Trailer	Anil	4
Updating the Task List and Timeline	All	20
Updating the Wiki Page	All	1
Demo Day Presentation	All	12
Presentation Slides (Final Release Slides)	All	6