Spyral Narwhals - "SoundMaze"

A survival game engine designed for the visually impaired.

Garreth Nedved, Gerritt Dorland, Jeremy Nicholson, Ryan Mann

PURPOSE

Using the Unity engine, create a game that is equally accessible to both the visually and non-visually impaired.
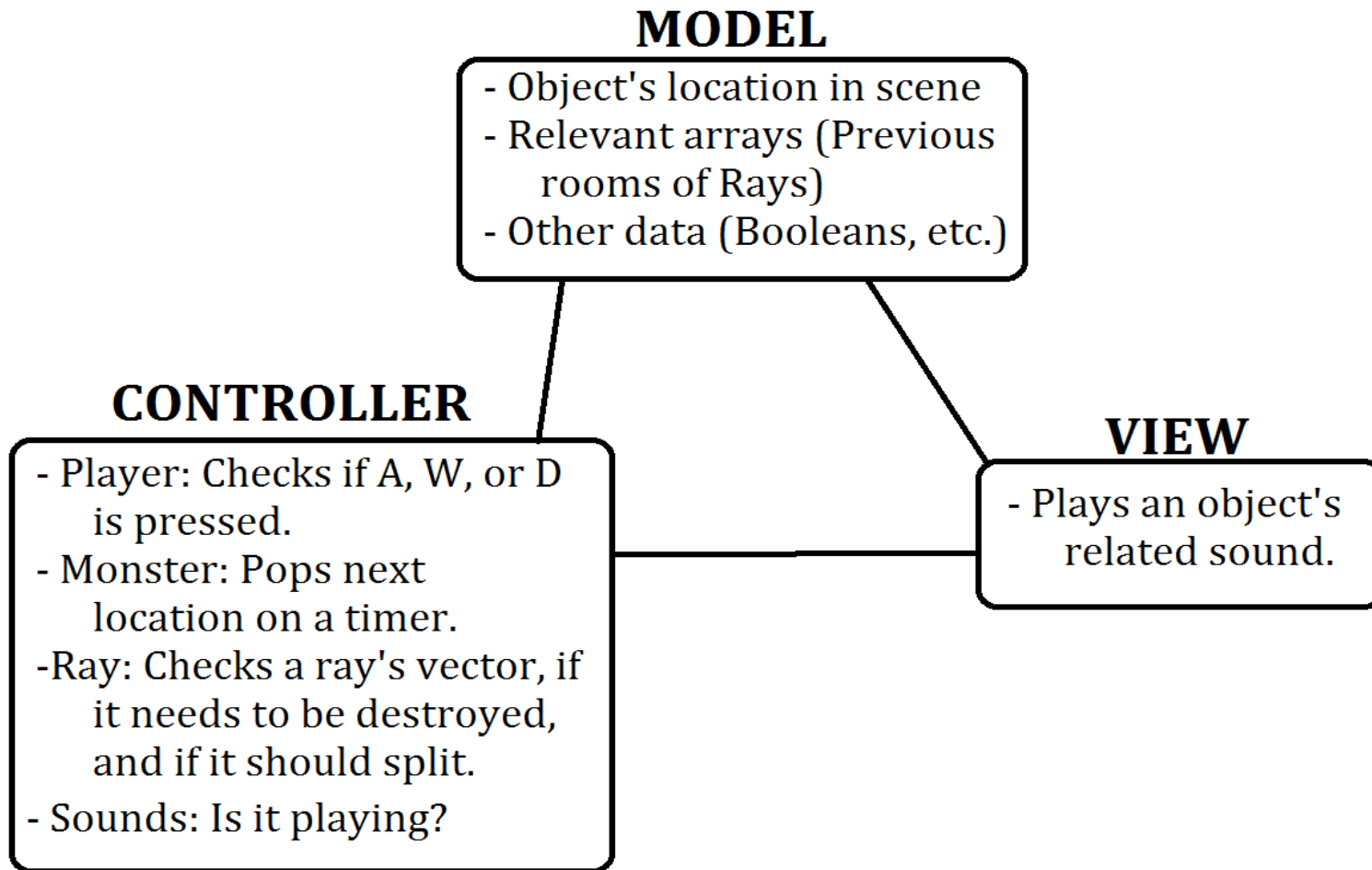
- Use only sounds to successfully guide players through levels.

- Make the game simple enough to be easily learned, but difficult enough to not be patronizing.

DESIGN

We used MVC:

- Due to how Unity functions, however, our MVC designs had to get mixed together in objects.
  - Was more useful making the components of an object (sounds, how it moves)
- View is pared down to only sounds played by an object
- Model was the largest part of our project
  - Saved the object translations from the Controller
  - All of our background data such as where an object is, its vector, and any associated arrays and data
- Controller handled how it moves
  - Player: User input controls
  - Monster: Stack of movements
  - Rays: Split or not? Destroy self or not?

OBJECT MVC DESIGN

MODEL
- Object's location in scene
- Relevant arrays (Previous rooms of Rays)
- Other data (Booleans, etc.)

CONTROLLER
- Player: Checks if A, W, or D is pressed.
- Monster: Pops next location on a timer.
- Ray: Checks a ray's vector, if it needs to be destroyed, and if it should split.
- Sounds: Is it playing?

VIEW
- Plays an object's related sound.

VERIFICATION

Verification for games is awkward!

- Unit testing is difficult – much of it ends up redundant in our case due to Unity's structure.

- The engine handles rendering and shading. As such, controls and visuals testing comes down to observation tests.

- What we do have thoroughly tested is our raycasting, room variables, and player movement

VERIFICATION

How We Test

- First, we have an external level that runs an integration test. This test mainly checks that player movement remains valid. Not part of the main program!

- Second, we run a quick unit test on our raycasting system in another level before loading the first level.

  - Quickly runs rays through various situations, and will throw an exception if a ray acts in an unexpected manner

TEAMWORK

To keep on schedule, we split into two groups: Design/Research and Coding.

- We needed people looking for solutions before we even reached a problem. Short on time.

- Design took ideas for solutions, and proved them on paper.

- Lead design then took the solutions to Coding and used Paired Programming (agile programming) to translate the designs into the game.

- Unity isn't very friendly with source control, so having one person manage the master was the easiest way.

WHAT WENT RIGHT?

- The way we broke down our team worked fairly well, considering Unity's limitations.

- Pre-designing systems allowed us to avoid huge hurdles before we even reached them.

- We designed very abstract systems before "game-ifying" the project, making everything abstract enough to quickly build new levels.

- Our systems work well with Unity, allowing us to use Unity's built-in level editor to add and remove elements (rooms, player location, goal point, etc.) simply by clicking and dragging it. No script editing is required – the game will still work.

WHAT WENT WRONG?

- We allowed ourselves to wait six whole weeks before abandoning the original project.

- We didn't account for unit testing until the very end.

- Often, we took a lot of Unity's framework for granted. We went into this project assuming Unity had soundcasting systems built in.

- We have more of a prototypical engine than a full game.

THE LAST TWO SPRINTS (11/25 – 12/3)

	Project	Story	Task	Start (NOV 26)	Day 7	Day 6	Day 5	Day 4	Day 3	Day 2	Day 1 (DEC 3)
1	Project	Story	Task	Start (NOV 26)	Day 7	Day 6	Day 5	Day 4	Day 3	Day 2	Day 1 (DEC 3)
2	Unity	Story 1	Finish Ray Caster	4	0	1	0	0	4	0	0
3	Unity	Story 2	Get Sound Object Placement working	5	0	0	0	0	1	2	2
4	Unity	Story 3	Research file I/O with Unity	2	1	1	0	0	0	0	0
5	Unity	Story 4	Implement File I/O, get reads working	2	0	0	2	0	0	0	0

THE LAST TWO SPRINTS (11/26 – 12/3)
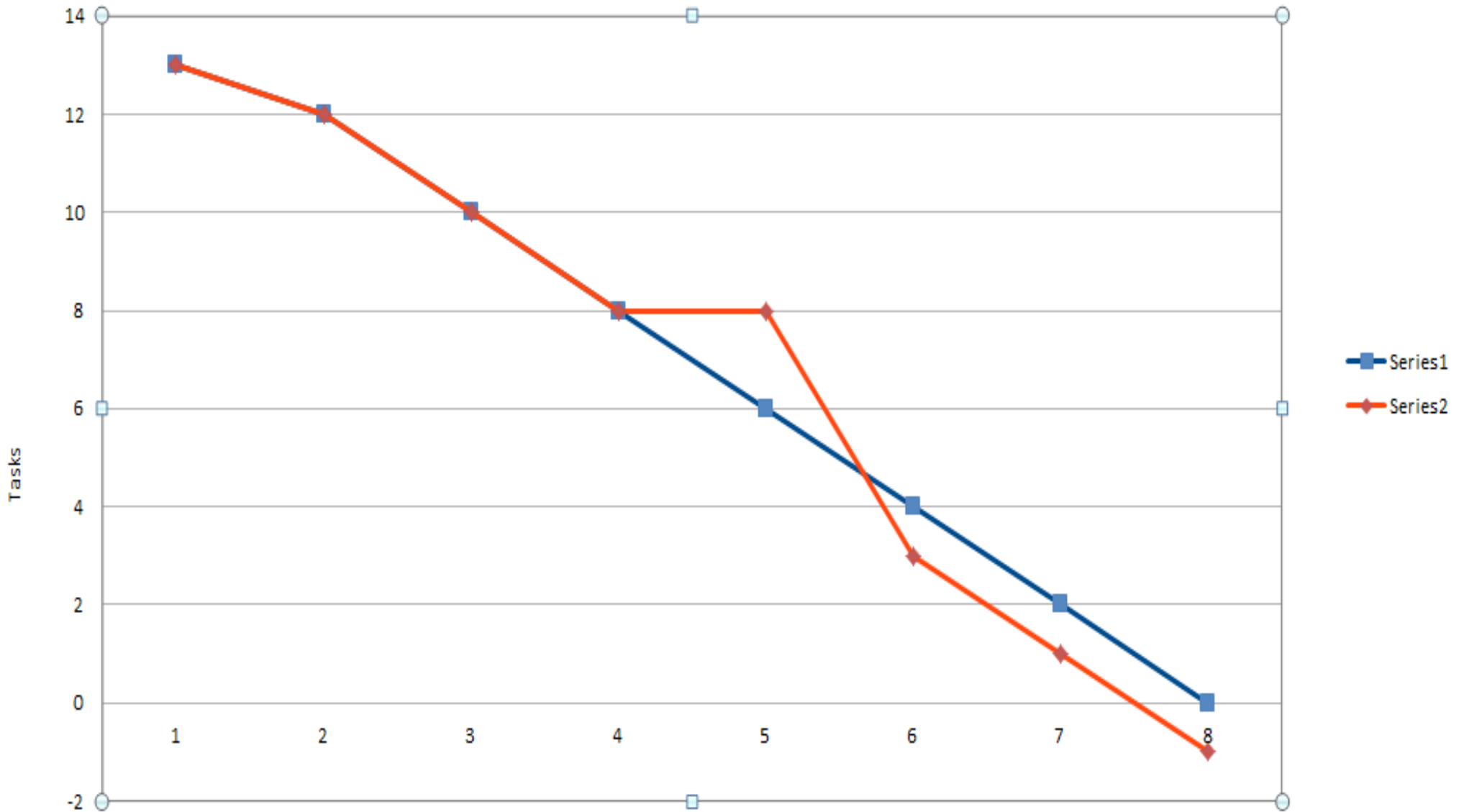
THE LAST TWO SPRINTS (12/4 – 12/11)

|	A	B	C	D	E	F	G	H	I	J	K
1	Project	Story	Task	Start (DEC 4)	Day 7	Day 6	Day 5	Day 4	Day 3	Day 2	Day 1 (DEC 11)
2	Unity	Story 1	Fix Sound Placement	4	1	1	1	4	4	0	0
3	Unity	Story 2	Implement Monster	4	0	0	0.5	0	4	0	0
4	Unity	Story 3	Add Exit Object	2	0	0	0	0	0	0	0
5	Unity	Story 4	Add SFX	3	0	1	0	0	0	1	0
6	Unity	Story 5	Add Unit Testing	7	0	1	0	1	4	0	0
7	Unity	Story 6	Make Presentation	4	0	0	0	0	4	2	0

THE LAST TWO SPRINTS (12/4 – 12/11)
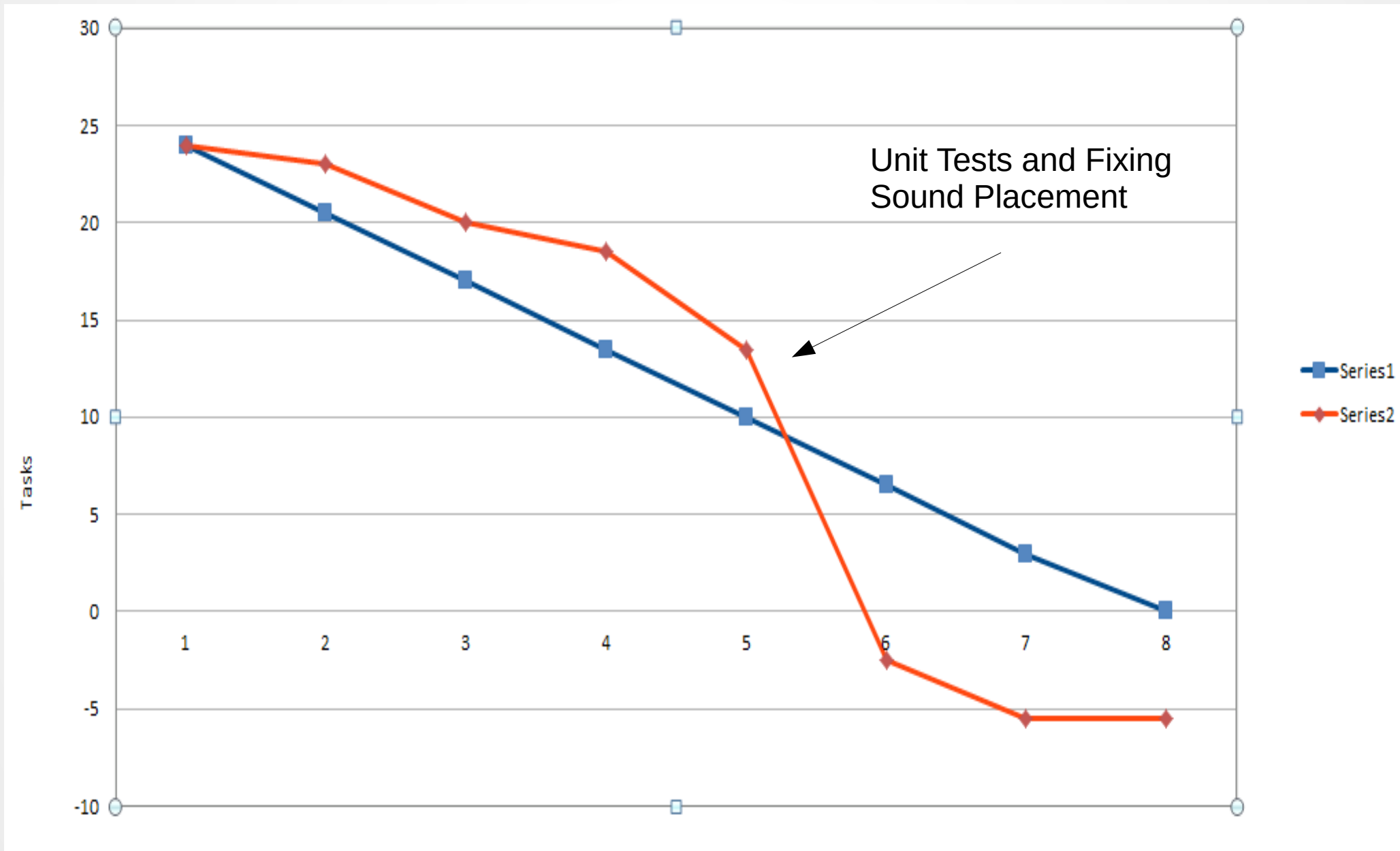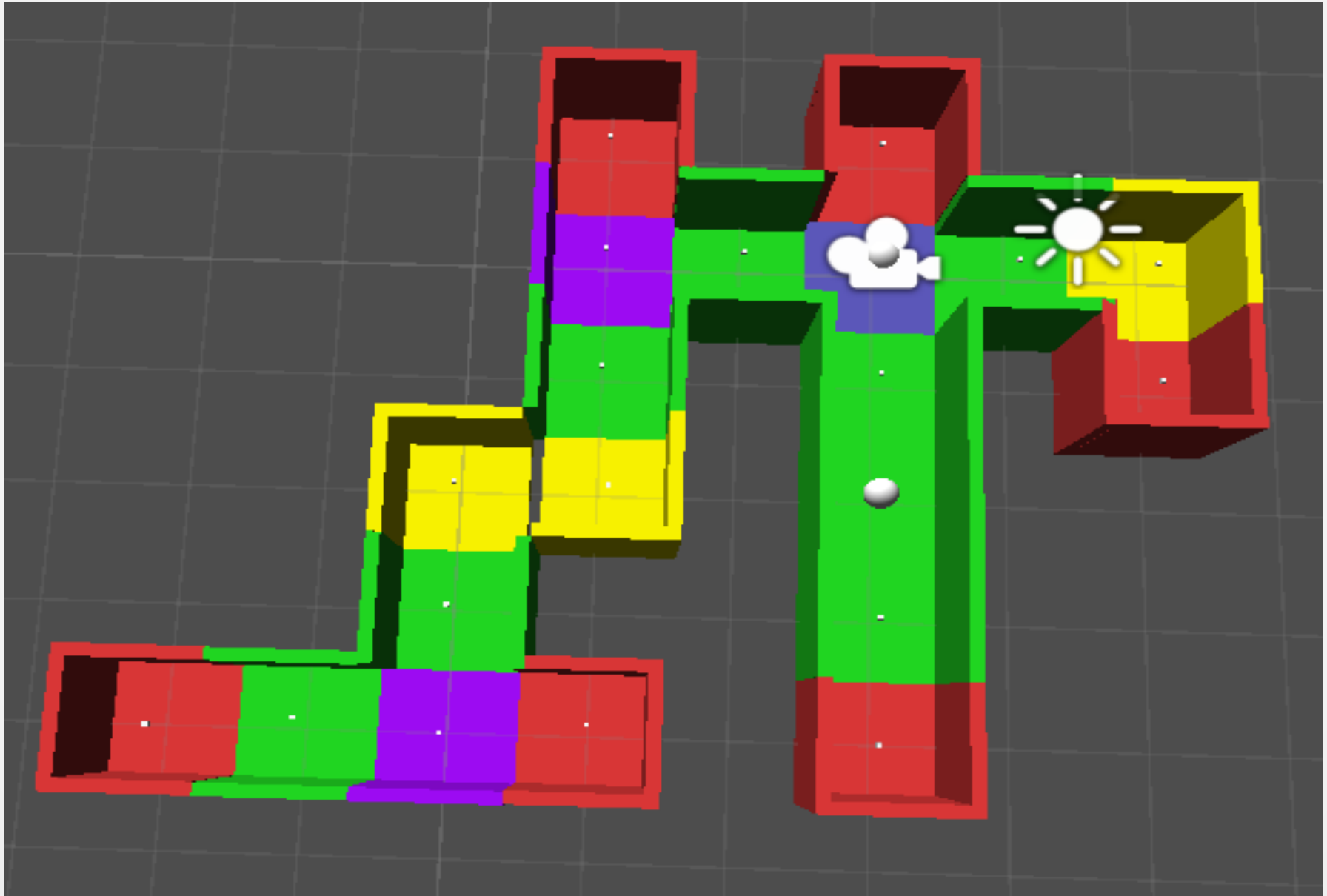
DEMO
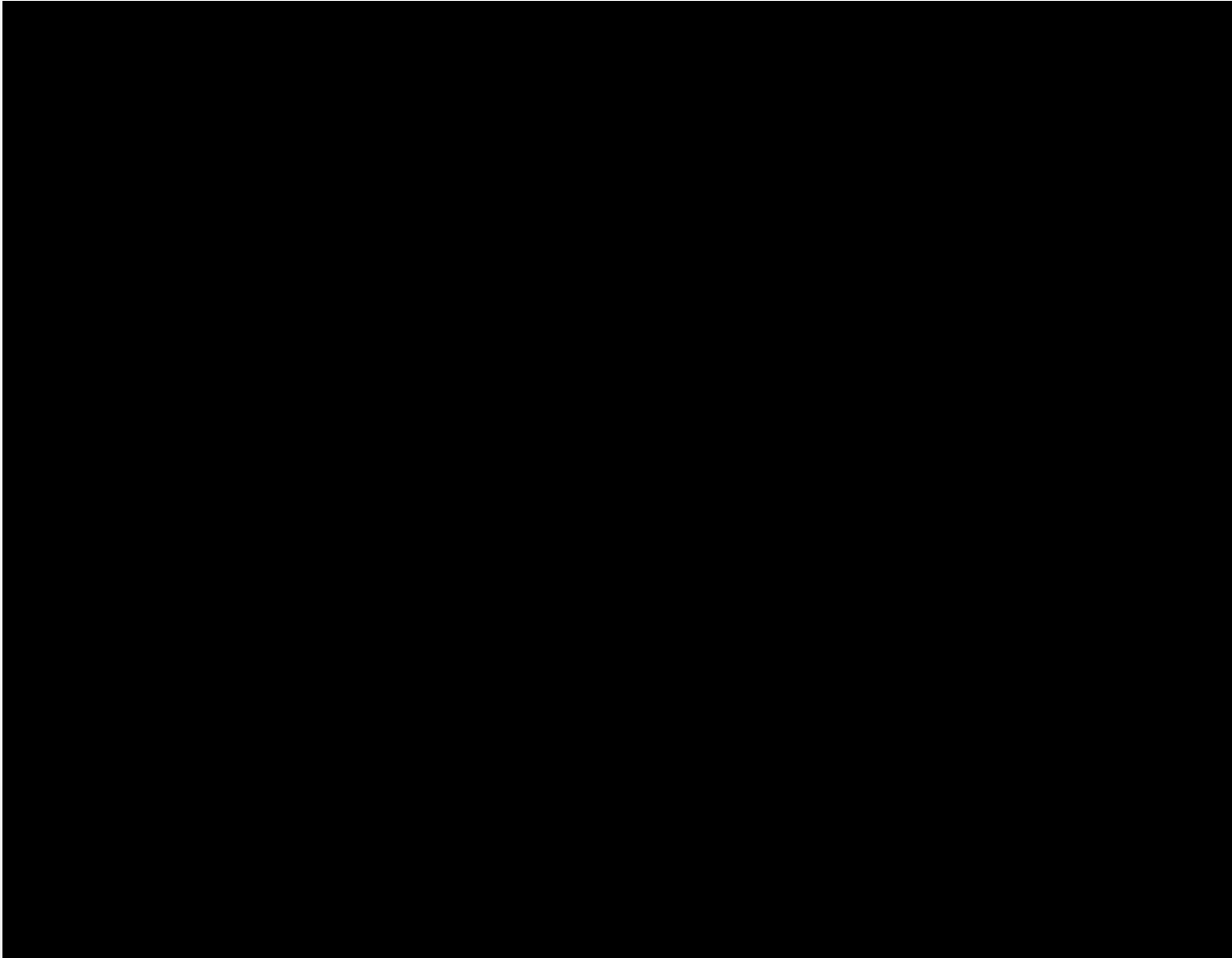
DEMO

Try out our demo game during our Demos on Monday!

CONCLUSION

A great start!

- The game's systems and backbone is functioning and intact.

- We built it to be easily extensible, meaning adding levels and content is as easy as dropping assets into a level.

- Did we achieve our goals? Sort of.
  - We'll find out how well our directionalized sound system works at the demos. We haven't had any QA yet!
  - We do have a very comprehensive prototype, however, that will be very easy to flesh out into a polished game.