

Invadem

**Design/Implementation Process**

This report outlines my process of designing and implementing the code for Invadem, a retro 2D shoot’em up.

Raunak Sitoula | rsit3870

**Milestone Submission**

**Setting Up**

The initial setup with Gradle was a little confusing, but once I finished setting it up and managed to successfully build it without errors, the milestone submission was a fairly straightforward process. I started off creating two classes:

* **Tank** – this class contains all (or almost all) data and logic related to the tank that players control in the game.
* **Invader** – this class is for the enemies the tank will be fighting off.

Next, I knew I would later be using some methods (such as collision detection on multiple classes; Tank, Invader, as well as the barriers and projectiles), so I created some interfaces for these:

* **IMoveable** – I created this interface with the intent of enforcing the same methods and properties on all objects in the game that move or can be moved. Within this, I also defined an enum, **Direction**, as I assumed I would need a more readable way of defining movement directions at some places in my code.
* **IHittable** – This interface houses a property for getting the health points of the object, and a method to “hit” and make the object receive damage.
* **IEntity** – This interface extends **IHittable** with a number of methods that are more specific to interactable entities in the game. It didn’t make sense, for example, for an interface called **IHittable** to have methods and properties related to things like its size, or whether or not it is “dead”. I decided to extend it into a single interface **IEntity** as I would still be using all of these things within the same context, and having these split up across multiple interfaces would be cumbersome and inefficient.

I also considered abstract/base classes, but decided against it as I had some entities who did not have the exact same requirements (Barriers and Tanks can both be hit, but Barriers can’t move), and inheriting from multiple classes is not possible. Therefore, interfaces were the obvious choice for me.

**Processing**

The next step was to get something visible up on the screen; I spent a bit of time figuring out processing and

**The Second Iteration**Taking the feedback from the evaluations into account, the second iteration went through a number of changes and improvements:

* The window size of the browser was considered and elements were scaled accordingly, while keeping the aspect ratio of the canvas fixed to 16:9.
* The bars buttons and instrument buttons were provided, to give users more control.
* Staves all started off at 6 bars so that the initial experience would generally be pleasant.

To get some feedback on these changes, evaluations were carried out one more time. The experience of users this time was generally quite positive, with only one main issue raised – the new settings buttons did not apply any visible changes until the next generation, and this was quite confusing for first time users.

**The Final Iteration**The final iteration saw the addition of the tooltip, which displayed an appropriate message at the bottom centre of the screen when hovering over a button.

**Concluding Notes**Beat.Gen improved quite a lot from the initial idea to what it is now, and the feedback from my peers and the evaluations were overwhelmingly positive. I’d like to thank everyone who helped with the evaluations for this, as well as the tutors for helpful advice and guidance; it’s been a fun assessment! Lastly, a special mention to Ableton[1], for providing the initial source of inspiration for Beat.Gen!