Team Galactic – Alien Invasion Sim

Contributors: Rocky Trinh, Zachary Powers, Joseph Lumpkin, Vincent Capra

For CS-417 / CS-505: Design Patterns

Submitted on: 12/5/2023

Professor: Chad Williams

# The Lore and Simulation

In this Alien Invasion Simulator our team has created, a grid is generated with 10 X 10 tiles, 30% of which become water. Then, resource tiles are spawned for: Uranium, Oil, Iron, and 1-Up Mushrooms. Humans and Martians are spawned in lopsided populations. Humans can reproduce but Martians cannot, thus their invasion force is a greater population than the Humans. The Martians attempt to gather resources in order to build a doomsday device! That way they can have all of Earth’s resources for themselves. They must collect 50 units of Oil, Iron, and Uranium to assemble their doomsday weapons. To spice things up, throughout the simulation random weather events indicated by the glowing border will spawn randomly for x number of years which buff or debuff the human attack modifier. After the Martians land, the scout units move in random directions until they find a resource and start to mine it. The Humans hunt the Martians by moving in random patterns, searching and try to eradicate them by attacking on sight (when they occupy a tile that is orthogonal (up, down, left right) to the human’s tile. The Humans know that Martian backup cannot arrive from Mars in time! If the humans can eradicate all of the Martians before the Martians can assemble their device, they can steal the plans for the doomsday device to defend against a future invasion!

A screenshot of a video game

Description automatically generated

Figure 1: Early game

A screenshot of a video game

Description automatically generated

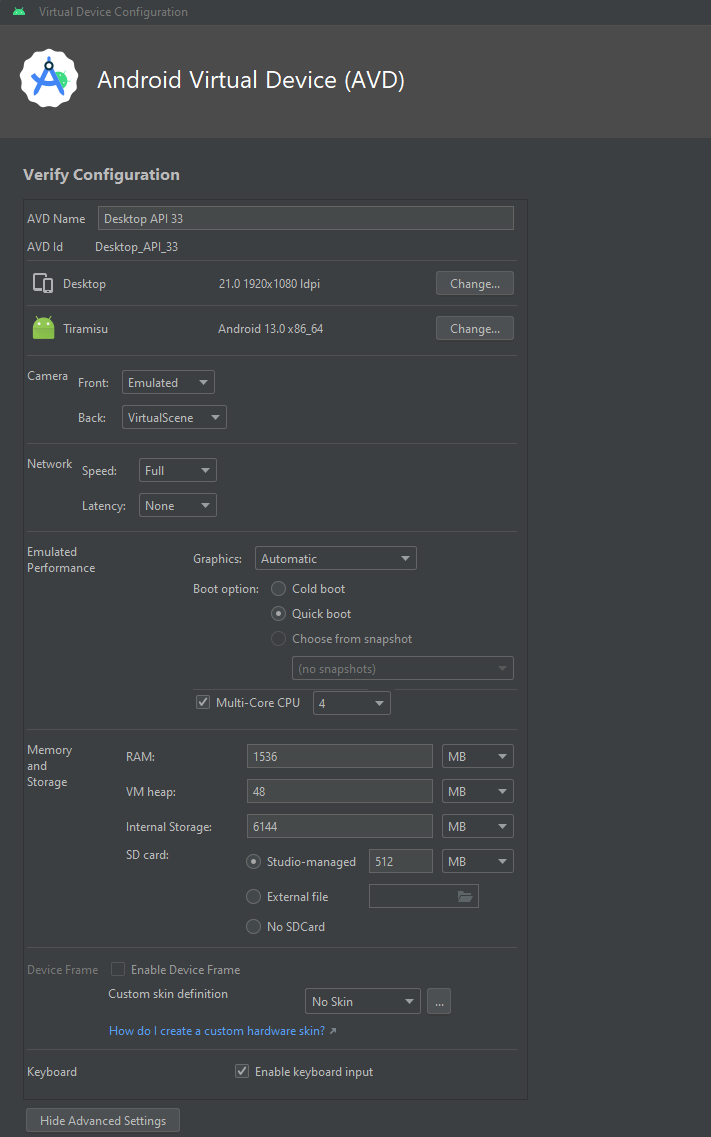
Figure 2: Mid Game

A screenshot of a video game

Description automatically generated

Figure 3: Humans Win!

# How to Configure and Run:

* Have an installed version of Android development studio (Android Studio Giraffe | 2022.3.1 Patch 2 used for this project’s dev/testing)
* Suggest creating an emulator with the following settings:
* 
* To avoid errors, you need to separately get the following gif “earth\_rotation.gif”:

<https://myccsu-my.sharepoint.com/personal/joseph_lumpkin_my_ccsu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fjoseph%5Flumpkin%5Fmy%5Fccsu%5Fedu%2FDocuments%2FCCSU%20FALL%202023%20CS417%20Transfer&ga=1>

Then the file needs to be manually placed in the project’s res > drawable directory.

The problem was that this graphic was too large for GitHub.

Once these steps are followed, you should be able to just run the application. No other input is necessary. You can advance the years manually until the simulation reaches a conclusion.

# Patterns Used

## Vincent’s Patterns

Abstract Factory – Implemented by Vincent Capra includes the following participants:

• Abstract Factory Class – AbsLifeFormFactory

• Concrete Factory Class – LifeFormFactory

• Abstract Product class – Lifeform

• Concrete Product Classes – Human, Xenomorph, Martian, Vulcan, Saiyan

Template – Implemented by Vincent Capra includes the following participants:

• Abstract Class – LifeForm

• Concrete Class – Human, Xenomorph, Martian, Vulcan, Saiyan

NOTE: In the end we only had enough time to implement a functional project with just the Martian Alien class. As noted in the UML, the gather() method holds the template:

gather(){  
 getNeighboringTerrainTileReferences() //same  
 getNeighboringResourceTileReferences() //same  
 checkForApplicableResources(){ //differs  
 mine() //differs  
 move() //differs  
}}

Composite – Implemented by Vincent Capra, includes the following participants:

• Component/Leaf Class – Lifeform (can be extended later/ broken down)

• Composite Class – LifeFormComposite

• Client – Main (this is the idea currently, but subject to change pending further imeplementation)

NOTE: I left this code in. It was stubbed out for implementation at the end of sprint 2, but we made a design decision to only have 1 Lifeform Object per tile and instead implement a “populationCount” int variable in the LifeForm class in lieu of managing multiple lifeform objects within a tile as a composite. This was the CORRECT decision given the time restraints. We spent days debugging the single objects’ movements, attacks, etc as it was. I’m almost positive that having multiple objects per tile would have complicated the project into an unmanageable state. However, I therefore have no actual functional implementation of this pattern.

Singleton – Implemented by Vincent Capra, includes:

• Singleton – Grid

Enum – Implemented by Vincent Capra includes:

• ResouceTile > resourceType (WATER, IRON, OIL, URANIUM, ONEUP)

Strategy – Implemented by Vincent Capra includes:

• Strategy: ReproduceStrategy

• Concrete Strategy A: HumanBaseReproductionStrat

• Concrete Strategy B: OneUpReproductionStrat

• Context: Human.mine()

## Joseph’s Patterns

Strategy – Implemented by Joseph Lumpkin includes the following participants:

• Interface – ProgressibleLifeForm

• Implementation: A hook to call all LifeForm activities as time progresses.

Observer – Implemented by Joseph Lumpkin includes the following participants:

• Subject: MutableLiveData<Integer>

• Concrete Subject: Grid.mYear

• Observer: java.util.Observer

• Concrete Observer: mYearObserver

The mYear object is watched for changes and alerts the MainActivity.updateDisplay(int) method to refresh what is shown to the user.

Iterator – Implemented by Joseph Lumpkin includes the following participants:

• GridView: Displays the information to the user as a GUI representation.

• Grid: Contains the information

• GridAdapter: Is the implementation of the interface called while looping over tiles and drawing the information to the screen.

## Zachary’s Patterns

Singleton – Implemented by Zack Powers, includes the following participants:

• Singleton Class – EventLogger

Memento– Implemented by Zack Powers, includes the following participants:

• Originator Class – Grid

• Caretaker Class – GridCaretaker

• Memento Class – GridMemento

Observer – Implemented by Zack Powers, includes the following participants:

• Publisher – Grid

• EventManager - EventManager

• EventListener - EventListener

• Subscriber - LifeForm

Prototype– Implemented by Zack Powers, includes the following participants:

• Base Prototype – LifeForm

• Concrete Prototype – Human, Martian, Saiyan, Vulcan, Xenomorph

Iterator – Implemented by Zack Powers, includes the following participants:

• Outer class/iterable object – Grid

• Inner class/iterator - GridTileIterator

• Iterator Interface – Iterator

Visitor – Implemented by Zack Powers, includes the following participants:

• Visitor Interface - TileVisitor

• Concrete Visitors - CloneTileVisitor

• Visitable Element- Tile

• Concrete Visitable Element - ResourceTile, TerrainTile

## Rocky’s Patterns

Iterator – Implemented by Rocky Trinh, includes the following participants:

• Iterator – Grid:progressLifeForms(). Iterates over the array of LifeForms and progresses them through one interval of time.

Enumeration - Implemented by Rocky Trinh, includes the following participants:

• Enumeration class - BuffDebuffTypes

• Tile class - applyBuffDebuff, removeBuffDebuff, getBuffDebuffValue

Strategy - Implemented by Rocky Trinh includes:

• Strategy: WeatherStrategy

• Concrete Strategy A: BlizzardWeatherStrategy class

• Concrete Strategy B: ClearWeatherStrategy class

• Concrete Strategy C: DroughtWeatherStrategy class

• Concrete Strategy D: FloodingWeatherStrategy class

• Context: WeatherContext

• setWeatherStrategy(), applyWeatherStrategy(), applyWeatherToGrid()

• WeatherStrategy interface

• applyWeatherEffect, getDuration, setDuration()

• Tile class

• mWeatherFlag(), setWeatherFlag(), getWeatherFlag()