Lab Report - Spy Game

By: Broncodes

Nick Huiting

Jose Rodriguez

Michael Ackerman

Thanh Doan

Tenzin Tashitsang

Dennis Jimenez

CS 141

Prof. Erwin Rodríguez

1 December, 2017

Introduction

The goal of this assignment was to use Java to make a text-based game as a team. We are the Broncodes, members include: Nick Huiting, Jose Rodriguez, Michael Ackerman, Thanh Doan, Tenzin Tashitsang and Dennis Jimenez.

Problem Description

This project is a turn-based game that satisfies these criterias:

- The player, six ninjas, one suitcase and three power-ups are spawn in a 9x9 building
- The room is dark, the player has a one-tile-radius vision
- The player has 3 lives.
- The player can shoot or move each turn as well as look once every turn.
- The player is able to use three available power ups in the building (Ammo, Invincibility, Radar).
- Have the enemies randomly move one space after each player turn.
- The player can kill an enemy if he/she has ammo and shoots toward the enemy.
- An enemy can kill the player when adjacent to the player.
- The building has 9 rooms, one of which has the briefcase.
- Running out of lives means you lose, finding and retrieving the briefcase means you win.

Approach to Solution (Design, Architecture)

The goal of this project was to create a robust yet expandable architecture that would allow for the possibility of multiple user interfaces and easily permit additions of other types of game objects and mechanics. To that end, the game engine was loosely coupled from the user interface. The game engine was passed an interface with minimal methods so that all input processing, game logic, and state changes would be done at the engine level. The UI would be tasked with prompting the user for input and displaying game data upon the engine's request. This way, both a text-based and graphical user interface would be made possible.

Another approach to make the program expandable was to implement an inheritance structure to allow for new types of game objects, power ups, or other items to be added to the game very easily. All objects in the game inherit a superclass, which declares abstract methods and holds general data pertinent to all objects in play. The Grid and sets of objects within each cell would only need to operate with respect to this superclass, and all descendent classes would be processed at the engine level, further consolidating game logic and increasing the reusability of the container objects. Through this design, adding a new type of power up, for instance would be trivial, and allow for faster development.

Additionally, to add a level of order to the game processing the engine was implemented as finite state machine. This meant that processing input would be state-dependent, simplifying the debug process and ensuring that if a fringe case occurred, the game would never enter an unknown state. This also allowed for more readable code, as processing by state naturally

separated code into methods. Additionally, this allows for a more expandable game because introducing a new game mode or user option would only involve implementing a new state.

We also created a class to hold game constants and used those values in processing in the engine. This helped to make the code more manageable and let us easily change values if necessary. This would make implementing a difficulty system trivial.

Through these design choices, game implementation went smoothly and required very little redesigning in the development process.

Discussion of Implementation

The codebase of the project was divided up into 4 sub packages along with 7 top level files. The subpackages are Engine, UI, Exceptions, and GameObjects. Work generally started with the initial creation of the classes with very little implementation. This was done to achieve the general structure of the project before committing to any implementation that may be underdeveloped or under planned. This would also prevent massive rewrites. This became the first milestone of the project. The second milestone involved implementation of the grid along with some simple, low level classes such as the gun. This was done because the grid is the most important stateful entity that would be managed by the GameEngine. Along the way, some of the simpler classes were implemented as well such as the GameObjects along with its subclasses. These classes were low hanging fruit and allowed us to build some momentum towards the project. Next, we worked on the movement of the GameObjects in the grid. By this point, a good portion of the UI and the GameEngine was completed, as they are required in order to do a few test runs of the program. Lastly, we did some code cleanup, documented all the classes and their

methods, and wrapped up small pieces of missing functionality. Further details of the implementation of each class are defined below.

For source control, we decided to use git and host on GitHub. This made collaborating on the project easy to track and prevented a lot of conflict issues. It was also a good introduction to git and GitHub for some members of the team.

Engine Package

This Package contains the following classes:

GameEngine

The GameEngine is the powerhouse behind this project. It takes

GameState

GameState is a simple enum that gets passed to the UI from the GameEngine in order for the UI to know what to output to the user. Enumerations for GameState are as follows: Menu, Saving, Loading, Playing, Moving, Shooting, PlayingAfterLook, Looking, Dead, Victory, Quit. Based on one of these, the UI will output the correct prompt to the user and return the user input back to the GameEngine.

GameSave

This GameSave class provides the user with the functionality to save games. This class implements Serializable and contains most of the attributes of the GameEngine class with the exception of the UI. While one could theoretically serialize the entire GameEngine, we decided against it because it would also serialize the UI instance, preventing one from starting a game on the command line and loading in a gui. Making the GameEngine serializable would also serialize all the it's methods, which is

unnecessary. All state is kept within the grid along with helper state objects such as GameState, making implementing a GameSave class quite easy. This class contains two static functions, saveGametoFile and loadGameSaveFromFile to assist with saving and loading respectively. GameSave also has setters and getters for its attributes, allowing for the creation and modification of GameSaves as well the creation of new GameEngine instances.

GameTurnResult

GameTurnResult in some ways serves a similar functionality to GameState. It is passed to the UI from the GameEngine, however it contains a bit more information. It contains all the data that would be relevant after a game turn. This includes the string representation of the grid in order to be printed, the number of lives the player as left, the current status of the player (defined by PlayerStatus below), as well as a UICommand (also defined further below) which tells the UI what to output. This class contains a constructor that is used by the GameEngine to create this object, as well as getters for the attributes mentioned above.

PlayerStatus

An immutable class that contains the player information resulting from the end of a turn. It is an attribute of GameTurnResult. Its attributes include whether the player as radar, whether the player has invincibility, whether the player has ammo, and the number of invincible turns left. This class is used to pass the above data to the UI in order to be displayed to the user.

UI Package

This package contains all relevant files pertaining to the user interface. Contains an interface that is to be implemented, a text-based user interface class, and a helper user interface command class.

IGameUI

The interface with which the engine will communicate information to the user, and receive input. This interface provides minimal methods, leaving all of the implementation up to a separate class.

UI

The UI is a text-based user interface, providing a means of inputting data and displaying textual information. This is used by the engine to get and send information.

UICommand

A command specifying how the UI should display information. This is passed with a result at the end of each frame by the engine. This can be used to bypass displaying anything for a frame, displaying a complex object, or just displaying an error message that resulted from a processing failure during input processing in the finished frame.

GameObjects Package

This package contains all classes that are relevant to the GameObjects, which are objects that can be stored in the grid and contain a position.

ActiveAgent

Extends GameObject and implements Serializable. It is also the parent of Enemy and Player. Contains a move function to change the position. Also overrides getPriority.

ActiveAgents get a higher priority when printing. (Ex: if an enemy occupies the same cell as a power up, the enemy is what gets printed).

Enemy

Extends ActiveAgent. Inherits all functionality of ActiveAgent and GameObject. Overrides getSymbol, as Enemies are displayed to the user as 'X'.

GameObject

The abstract superclass for all game objects. It only contains 1 field attribute which is the Position. It's methods include a getter for the position, a constructor, an abstract getSymbol method that is to be implemented by the subclasses. Each subclass is to define how they want to be printed. The abstract method also has arguments for debug (boolean) and radar (boolean). The subclass will also define how they are to be printed depending on the powerup (radar) or if debug is turned on. One last method, which should also be overridden, is getPriority, which returns the visibility priority. The visibility priority determines which GameObject gets printed if two GameObjects occupy the same cell (ex enemy on the same cell as a power up).

GameObjectSet

GameObjectSet is how two GameObjects occupying the same position is implemented. The grid is actually made up of GameObjectSets, defined as GameObjectSet[][] grid. This object contains 2 fields: 1 GameObject array of size 2, and a counter of how many objects the set currently has. This class contains helpers to add and remove GameObjects, useful for when GameObjects such as enemies move into or a

leave a cell. Also contains a getSymbol function that prints out the GameObject with the highest visibility priority in it.

Player

Extends ActiveAgent.

PowerUp

A Power up that can be placed on the grid. This has a type to determine how it should be used in the event it is picked up by the player.

PowerUpType

Helper enum to determine the type of PowerUp. The enumerations are as follows: Radar, Ammo, Invincibility. This enum is a private member of PowerUp.

Room

Room class which extends GameObject and implements Serializable. Inherits all functionality of GameObject and also adds a hasBriefcase attribute (boolean). Overrides getPriority to determine visibility priority. Overrides getSymbol as well to allow the room to be printed. Returns '?' if there's a briefcase, otherwise returns '@'. Also contains setters and getters for hasBriefcase.

VisibilityPriority

Provides priority with an integer backing value that allows for comparison. Used by game objects to determine the render order in a GameObjectSet if there are two objects occupying the same cell.

Exceptions Package

Package that contains all the custom exceptions for the game.

GameStateException

A custom exception thrown if the game enters an unexpected state and processing is impossible.

PositionException

A custom exception thrown if a Position is invalid.

Other Files

The following files are not grouped into their own subpackage, therefore this sit at the top level of src folder. In many cases, these are just helper classes or they don't categorically belong in one of the subpackages above.

CardinalDirection

Represents a cardinal direction (Up, Down, Left, Right), and used to determine directions that a player can look, move, and shoot in the grid.

Constants

Helper class that simply contains static immutable constants for the game. These constants include number of rows and columns for the grid, number of enemies, number of rooms, the distance a player can look, number of invincible turns for the invincibility power up, and the minimum distance between a player and an enemy at the start of the game.

Grid

A wrapper around a 2D array that provides methods for adding, removing, moving, and displaying its contents. This is used as the board of the game, and can be expanded to any size.

Gun

Simple Gun class that acts as a wrapper for ammo. It manages the the ammo of ammo available and includes a setter and getter for ammo.

Main

Main point of entry to launch the application. The UI is created in the main method and passed to the engine on start.

Position

A helper class to deal with positions. Simply a wrapper around (int x, int y). This class is immutable, therefore once a Position it cannot be changed. Contains getters for x and y as well as the following helper functions: posEquals to test whether two positions are equal, toString to convert the position into a string, isAdjacent to determine whether another location is adjacent to the current position, and changeX and changeY to return a new X position with the same Y and a new Y position with the same X respectively.

Utilities

A helper class that only contains static helper methods. These helper methods are: priorityGreater, which determines which priority is greater in regards to visibility and which GameObject should be printed when 2 GameObjects reside in the same cell, positionLooked, which determines whether a given position is a space being looked at by a player (assists with outputing), and getRandomValidPosition which is used in the spawning of ninjas.

Testing Data

The following is a table of use cases and testing data.

Scenario	Expected Behavior	Actual Behavior	Passed
Press new game	Enter a new game	Enter a new game	**TRUE**
Press enter without typing anything	Invalid input message and reprompt	Invalid input message and reprompt	**TRUE**
Press quit	Exits program	Exits program	**TRUE**
Press load save	Prompts for a filename	Prompts for a filename	**TRUE**
Toggle debug	Shows and hides players, power ups, briefcase	Shows and hides players, power ups, briefcase	**TRUE**
Look into a wall	Nothing happens, look is consumed	Nothing happens, look is consumed	**TRUE**
Move into a wall	Invalid move, reprompt	Invalid move, reprompt	**TRUE**
Try to look after looking	Display an error, don't allow user to look	Display an error, don't allow user to look	**TRUE**
Shoot into a wall.	Invalid input, reprompt	Invalid input, reprompt	**TRUE**
Move or shoot.	Ninjas move	Ninjas move	**TRUE**
Shoot in the direction of a ninja	Ninja is removed from board.	Ninja is removed from board.	**TRUE**
Ninja or player moves on top of power up	Ninja or player symbol shown in cell	Ninja or player symbol shown in cell	**TRUE**

Try to move into room cell from left, right, bottom	Invalid move, reprompt	Invalid move, reprompt	**TRUE**
Try to move into room from top	Enter room.	Enter room.	**TRUE**
Player enters room	Player is rendered on top of room	Player is rendered on top of room	**TRUE**
Player enters room with a briefcase	Player wins	Player wins	**TRUE**
Player enters room with no briefcase	Nothing happens	Nothing happens	**TRUE**
Player in room tries to exit from bottom, left, right	Invalid move, reprompt	Invalid move, reprompt	**TRUE**
Player in room tries to exit from top	Player exits the room	Player exits the room	**TRUE**
Player looks in a direction that has at least 2 spaces ahead.	The cell 2 cells away from the player in the direction is revealed	The cell 2 cells away from the player in the direction is revealed	**TRUE**
Player wins	Menu to restart, load, or quit	Menu to restart, load, or quit	**TRUE**

Player starts a new game	Ninjas spawn at least 3 spaces away from player	Ninjas spawn at least 3 spaces away from player	**TRUE**
Player steps on radar power up	Player sees which room the briefcase, power up is consumed	Player sees which room the briefcase, power up is consumed	**TRUE**
Player steps on ammo power up with no ammo	Ammo is returned to full, power up is consumed	Ammo is returned to full, power up is consumed	**TRUE**
Player steps on ammo power up with ammo	Nothing happens, power up is consumed	Nothing happens, power up is consumed	**TRUE**
Player steps on invincibility power up	Player is invincible for 5 turns	Player is invincible for 5 turns	**TRUE**
Player moves while invincible	Invincibility decreases by one	Invincibility decreases by one	**TRUE**
Player shoots while invincible	Invincibility decreases by one	Invincibility decreases by one	**TRUE**
Toggle debug off after looking	Grid still displays with the look square revealed	Grid still displays with the look square revealed	**TRUE**
Invincibility decreases to 0	Player no longer invincible	Player no longer invincible	**TRUE**

Player hits ninja while not invincible	Player loses a life, game resets	Player loses a life, game resets	**TRUE**
Player hits ninja while invincible	Nothing happens	Nothing happens	**TRUE**
Player moves to space with ninja and power up	Player loses a life, game resets	Player loses a life, game resets	**TRUE**
Player moves in selected direction	Player is moved in the selected direction	Player is moved in the selected direction	**TRUE**
Player tries to enter room from the bottom left or right	Invalid move	Invalid move	**TRUE**
Player inputs invalid file name	File error	File error	**TRUE**
Player loses all lives	Game Over and loads game start prompt	Game over and loads game start prompt	**TRUE**
Press Menu	Prompts menu	Prompts menu	**TRUE**
Press Help	Displays game description	Displays game description	**TRUE**
Press Move	Gives option for up,down,left or right	Gives option for up,down,left or right	**TRUE**
Player debugs a second time	Grid returns to hiding briefcase,power ups and ninjas	Grid returns to hiding briefcase,power ups and ninjas	**TRUE**

Player debugs and plays new game	Grid remains visible	Grid remains visible	**TRUE**
Player gives invalid direction to look	Invalid input. Please try again. reprompt	Invalid input. Please try again.reprompt	**TRUE**
Player gives invalid move direction	Invalid input. Please try again.reprompt	Invalid input. Please try again.reprompt	**TRUE**
Player shoots ninja with a ninja behind	Kills only the ninja in front	Kills only the ninja in front	**TRUE**
Player looks while invincible	Invincibility has same amount of turns left	Invincibility has same amount of turns left	**TRUE**
Player debugs and then lands on radar	nothing happens	nothing happens	**TRUE**
Player looks after debug	nothing happens	nothing happens	**TRUE**
Player enters empty room	Player is revealed over empty room and nothing happens	Player is revealed over empty room and nothing happens	**TRUE**
Player attempts to leave room from left, right of bottom	Invalid move, reprompt	Invalid move, reprompt	**TRUE**
Player shoots ninja through room	Shot hit! Ninja is removed	Shot hit! Ninja is removed	**TRUE**

Player spawns on power up	power up is hidden till player moves and isn't used	power up is hidden till player moves and isn't used	**TRUE**
Player shoots ninja through powerup	Shot hit! Ninja is removed	Shot hit! Ninja is removed	**TRUE**
Player debugs	Turn isn't consumed and grid is visible	Turn isn't consumed and grid is visible	**TRUE**
Player looks more than once per turn	You may only look once per turn	You may only look once per turn	**TRUE**
Player looks	Turn isn't consumed, ninjas don't move	Turn isn't consumed, ninjas don't move	**TRUE**
Player loses a life with no ammo	player is reset with ammo	player is reset with ammo	**TRUE**
Player looks at room	Briefcase isn't revealed	Briefcase isn't revealed	**TRUE**
Player misses shot	"Shot missed" ammo is consumed	"Shot missed" ammo is consumed	**TRUE**
Player attempts to move outside grid	Invalid move,reprompt	Invalid move, reprompt	**TRUE**
Player loses a life with radar enabled	radar is disabled	radar is disabled	**TRUE**

Player loses a life after killing a ninja	a new ninja takes its place	a new ninja takes its place	**TRUE**
Ninja is adjacent to player	Player loses a life	Player loses a life	**TRUE**
save with radar enabled	loads with radar enabled	loads with radar enabled	**TRUE**
load	prompts for save file name and loads file	prompts for save file name and loads file	**TRUE**
save with no ammo	loads with no ammo	loads with no ammo	**TRUE**
save with ammo	loads with ammo	loads with ammo	**TRUE**
saves with x invincibility left	loads with exact invincibility left	loads with exact invincibility left	**TRUE**
saves after killing ninja	loads with exact amount of saved ninjas	loads with exact amount of saved ninjas	**TRUE**
saves with x lives	loads with x lives	loads with x lives	**TRUE**
saves with radar disabled	loads with radar disabled	loads with radar disabled	**TRUE**
saves with invincibility disabled	loads with invincibility disabled	loads with invincibility disabled	**TRUE**
save with debug enabled	loads with debug enabled	loads with debug enabled	**TRUE**
save more than once, same name	loads overwritten save	loads multiple saves	**TRUE**

load non-existent save	gives file error	gives file error	**TRUE**
Invalid menu input	gives input error, try again	gives input error, try again	**TRUE**
push help multiples times	displays help multiple times	displays help multiple times	**TRUE**
save multiple times with different names	saves multiple files	saves multiple files	**TRUE**
look into a room without briefcase	nothing happens	nothing happens	**TRUE**
look into a room with a briefcase	nothing happens	nothing happens	**TRUE**
shoot a room that contains a briefcase	bullet goes through the room	bullet goes through the room	**TRUE**
turn off debug with radar on	radar is still active	radar is still active	**TRUE**
throughout game does ninja enter a room	never	never	**TRUE**
load saved file after closing and opening the program	loads the saved file	loads the saved file	**TRUE**
shoot a power up	goes through the power up	goes through the power up	**TRUE**

starting a new game after having debug on	debug is turned off	debug is turned off	**TRUE**
stepping on a ninja when invincible	shows player, no damage taken	shows player, no damage taken	**TRUE**
staying on an enemy while invincibility runs out	player should die	player dies	**TRUE**
player picks up ammo with invincibility	stays invincible and gets ammo	stays invincible and gets ammo	**TRUE**
player picks up radar with invincibility	stays invincible and gets radar	stays invincible and gets radar	**TRUE**
player picks up ammo with radar enabled	keep radar and get ammo	keep radar and get ammo	**TRUE**
player picks up invincibility with radar enabled	keep invincibility and get radar	keep invincibility and get radar	**TRUE**
invalid input during player choice	ask to input again	ask to input again	**TRUE**
load game after winning	loads game	loads game	**TRUE**
exit game after winning	exit game	exit game	**TRUE**
new game after winning	loads new game	loads new game	**TRUE**
load game after losing	loads game	loads game	**TRUE**
new game after losing	loads new game	loads new game	**TRUE**

exit game after losing	exit game	exit game	**TRUE**
entering a valid input with an invalid input after EX: 1 W	invalid input	says invalid input	**TRUE**

Conclusions

Over the course of this project we learned many things. We discovered communication is vital to creating a successful product and the more we communicate, the smoother the development process goes. We also learned that code organization using subpackages is very important, and helps reduce the chaos of too many classes in a single level. Additionally, the use of enums to keep track of states proved to be very useful and made the code very readable. Another thing we learned is to create code that is expandable and reusable, as that will allow for more succinct code and better organization of the code itself among files. Lastly, we learned the value of testing and writing code specifically for debugging in order to make the testing easier and more effective.

Suggestions for Improvements

One possible improvement would be to improve the debugging system. While the currently implemented debugging system is works sufficiently, having stronger debugging tools could make it easier to test certain edge cases that are reliant on the randomness of the ai. This was apparent when we were testing certain cases such as checking what happens when a powerup ends while you're on top of a ninja. With the included debugging tool, we had to

constantly save and reload until we were able to move in the same direction as the ninja. Having a debugging tool to control the ninja’s position and manually manipulate other game variables would've made testing edge cases we much easier.