SpyGame

Broncodes

Nick Huiting
Jose Rodriguez
Thanh Doan
Tenzin Tashitsang
Dennis Jimenez
Michael Ackerman

Problem Description

Create a turn-based game with the following mechanics:

- A spy type game where everything is dark except one tile around the player.
- The player has 3 lives.
- The player able to move in a 9x9 grid building.
- The player can shoot or move each turn as well as look once every turn.
- The player is able to use power ups that are in the building (Ammo, Invincibility, Radar).
- Randomly populate the building with 6 enemies and one of each power up.
- 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, can only be accessed from North.
- Running out of lives means you lose, finding and retrieving the briefcase means you win.

Approach To Solution

- Decoupled architecture with engine and UI separated.
  - UI is owned by engine. UI completely separate from game logic.
  - After processing, engine sends command to UI to display a result.
- Expandability
  - Support for any type of user interface: console, GUI.
  - Inheritance structure: Easy to add new power-up types or other game objects.
- Finite State Machine
  - Engine operates based on specific states. UI prompts and input processing depend on states.
- Easy to Change
  - All game values consolidated into one class

Implementation

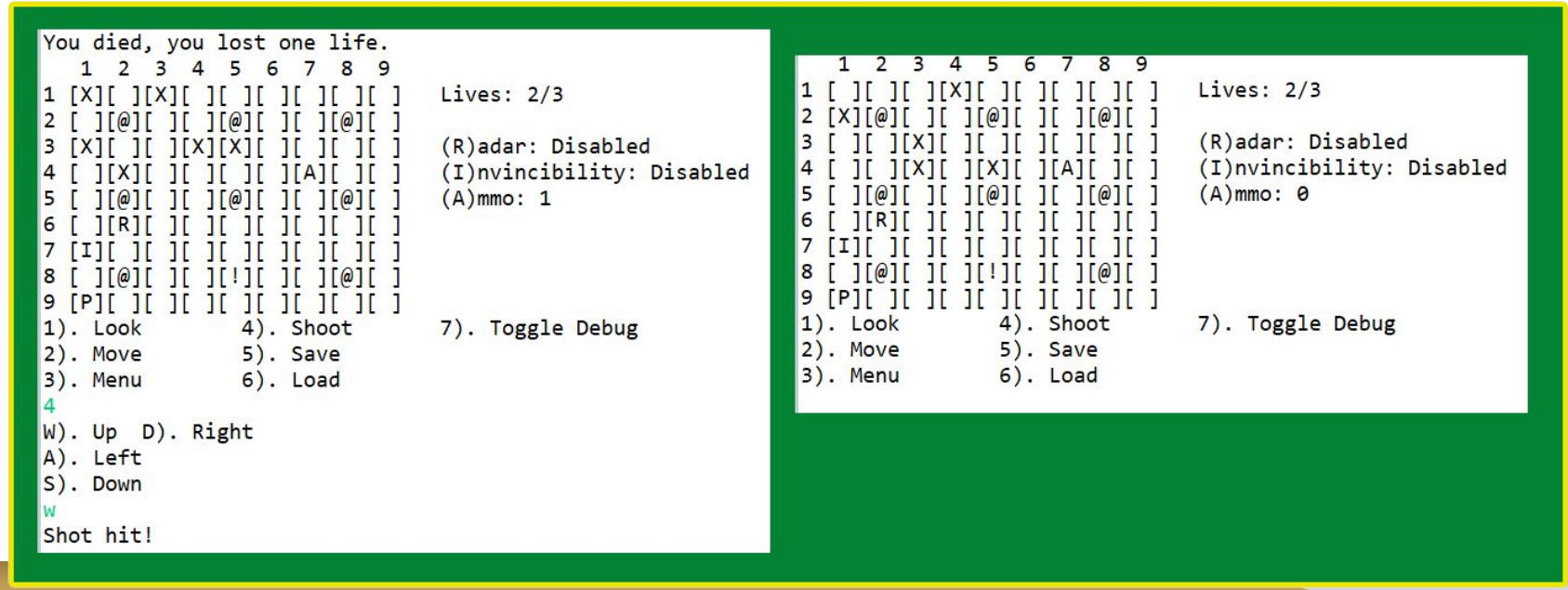
- 2 main sub-packages: Engine and UI. Engine contains all logic that handles the game state. This package includes not just the GameEngine, but also GameState, PlayerTurnResult, PlayerStatus, and GameSave classes. On the other hand, UI exclusively deals with user interaction. UI is instantiated on start and managed by the Engine. States and commands are passed to the UI in order for the UI to display data and prompt the user accordingly.
- 2 minor sub-packages: GameObjects and Exceptions. GameObjects groups all objects that can be stored in a grid (power ups, enemies) as well as their sub and super classes. Exceptions contains custom exception classes.
- 7 other files which include low level classes, such as the Grid and Gun, as well as helper classes with static members, such as Utilities and Constants.

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, powerups, briefcase	Shows and hides players, powerups, 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

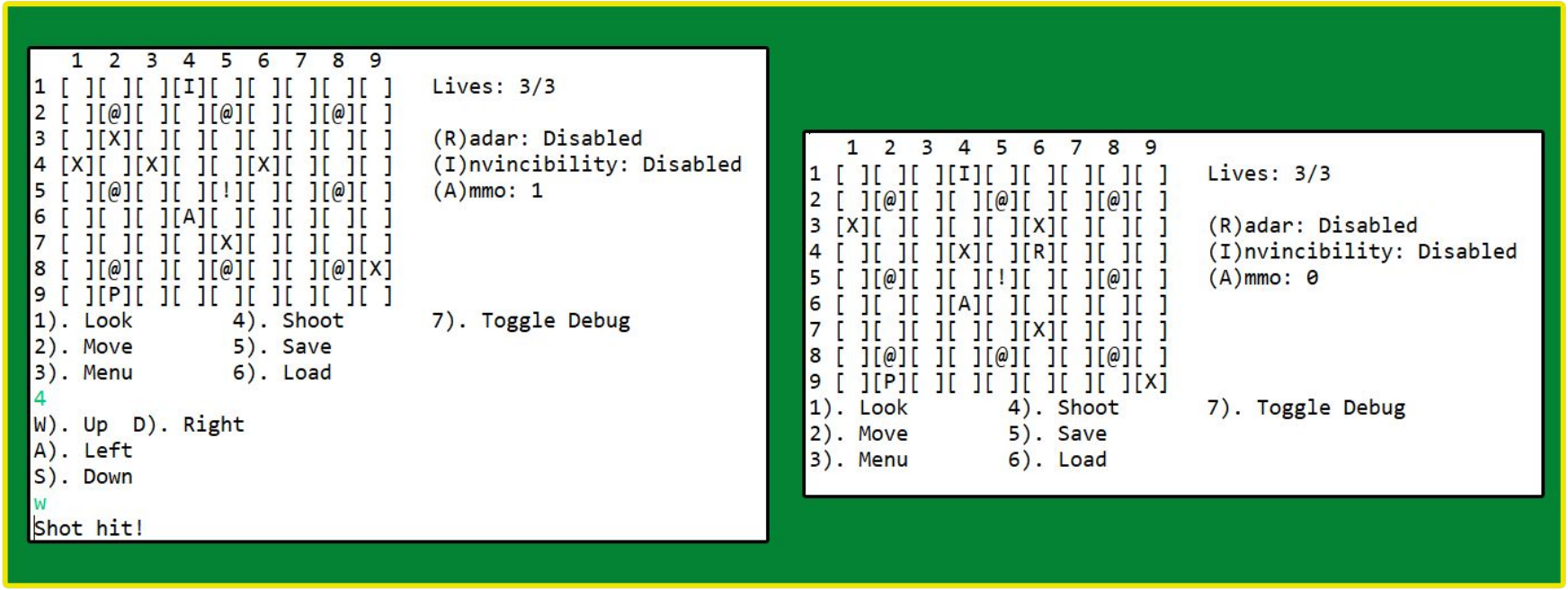
Testing Data

Scenario: Player shooting ninjas back to back.
Expected Behavior: Player should shoot and kill the first ninja it hits.
Actual Behavior: Player shoots and kills the first ninja it hits.



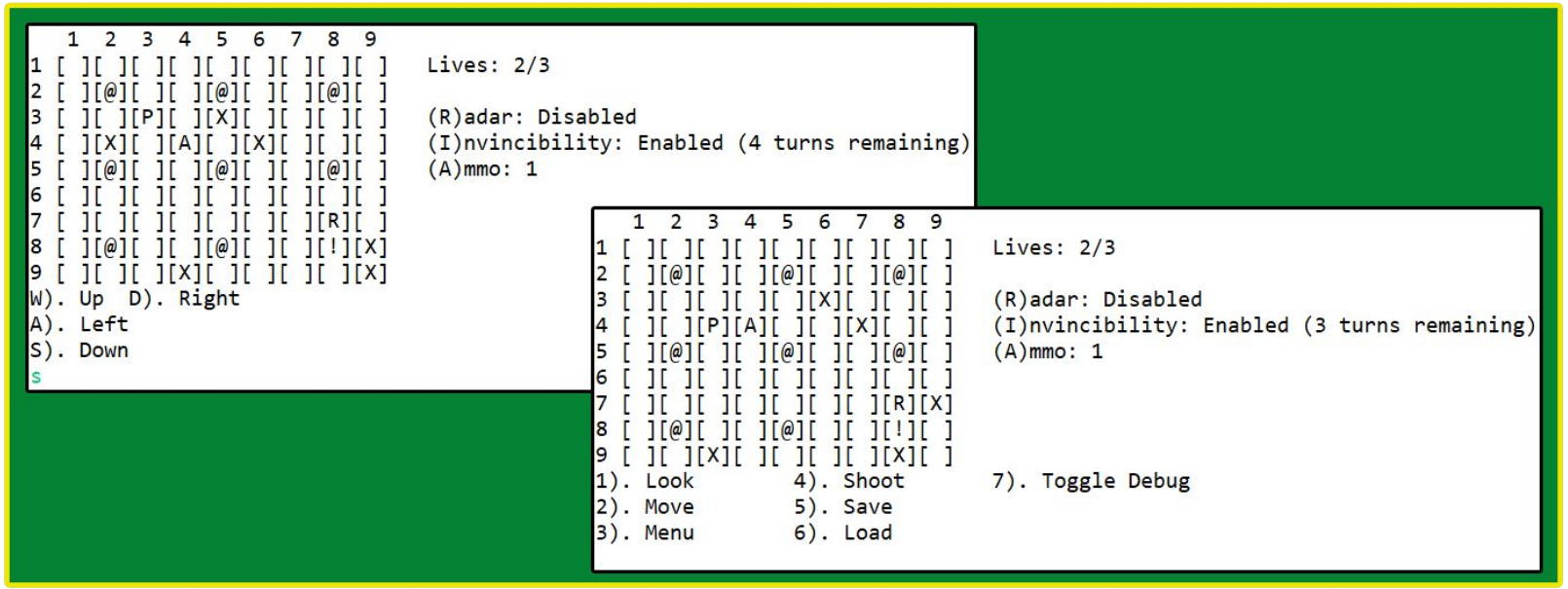
Testing Data

Scenario: Player shoots ninja while separated by rooms.
Expected Behavior: Bullet should go through rooms and hit the ninja.
Actual Behavior: Bullet goes through rooms and hits the ninja.



Testing Data

Scenario: Player with invincibility moves on top of a ninja.
Expected Behavior: Player shouldn't lose a life and Player symbol is shown in cell.
Actual Behavior: Player doesn't lose a life and Player symbol is shown in cell.



Conclusions

- Working with a team requires a lot of communication.
- The use of enums as constants for lists and flags.
- Code structure with the help of packages.
- Creating code with more reusability and expandability.
- How to test software and create testing data.

Suggestions for Improvement

- Communicate with teammates better.
  - Partitioning the work from the beginning to avoid confusion.
- Create a better debugging tool for testing difficult test cases.
  - Difficult to test with randomness.
    - Number the ninjas.
    - Control the ninjas/ freeze the ninjas.
    - Manually manipulate other game variables.

Suggestions for Improvement, Cont.

- Add different types of enemies.
- Make the rooms appear in random places while not letting them touch.
- Make some rooms have different power ups.
- Make stairs that allow the player to go up and down levels.
- Add different types of weapon that are randomly placed.
- Allow the player to hide 3 times per game so that enemies don’t run into him/her.
- Have random candles/ lights in the level that allow the player to see tiles around the light.
- Scoring system bases on time and items acquired.
- Randomly scatter intel (not the briefcase) that give bonus points.
- Save top three top scores.
- Let enemy steal the briefcase before the player.
- Add multiplayer component