2 Task

Please ensure you read all of the assessment carefully before you start working.

Your task is to write a MIPS program that implements a basic game, details of which are provided below. There are restrictions on which `syscall` functions you can use in your code. You are also required to write a short report detailing your approach to solving the problem.

I will test your game using the MARS simulator, specifically the one that is available through the 'SCC Lab' VM. Ensure you have tested your game on this platform before submitting.

Permitted `syscalls`

You are only permitted to use the following seven `syscalls` in your MIPS code. All other I/O and device interaction needs to be coded by hand. Note that the 'print string' `syscall` is only permitted for debugging purposes (see rule 1 below).

- Memory allocation (sbrk) (`v0 = 9`)
- Print string (`v0 = 4`) – debugging only
- Print integer (`v0 = 1`)
- Exit (`v0 = 10`)
- Set PRNG seed (`v0 = 40`)
- Generate random integer (`v0 = 41`)
- Generate random integer in range (`v0 = 42`)

2.1 The Game

The game consists of a simple walled environment (see fig. 1 for one such example). The player (shown as '`P`' in the below example) can move around the environment using the keyboard to collect different rewards (shown as '`R`' in the example).

```
Score: 25
#########
#   R   #
#       #
# P     #
#       #
#       #
#########
```

Figure 1: Example of a one possible game display, showing a 7-by-7 environment with one reward (`R`), one player (`P`), and the current score (`25`). Walls are represented by '#'.

The rules of the game are as follows:

1. The game must be presented to the user via the 'Display' device in MARS.[2]
2. The user should be able to interact with the game via the 'Keyboard' device in the lower part of the same window.
3. The user should be able to use the '`WASD`' keys to move the player around the environment.
4. Each collected reward should contribute 5 points to the player's score.
5. When a reward is collected, a new reward should appear in a different randomly-allocated location in the environment.
6. The score should be presented to the user at the top of the display in the form '`Score:  25`'.
7. If the player collides with a wall (i.e. tries to move into the wall), or if the player reaches 100 points, the game should end.
8. When the game ends, the display should be cleared and a message displayed saying '`GAME OVER`', along with the final score.

[2]this is available in MARS under the Tools > Keyboard and Display MMIO Simulator menu option

2.2 Game Extension

The second part of your task is to build an extension to the game. Be sure to make a copy of your implementation to part 1 (the basic game) and submit them separately so that I can test each part in isolation. The extension should be *one* of the following options:

1. Make the game two-player; Use the WASD keys for player 1 and the IJKL keys for player 2.
2. Give the player a speed; once the player starts moving, they should not be able to stop, only change direction.
3. Add an enemy; the enemy should move every time the player does, and should aim to prevent the player reaching the reward (colliding with the enemy is the same as with a wall).

2.3 Documentation

You need to write a short report on your game. This should describe the problems you encountered, your high-level approach to the problems, and provide detail about how you solved them. Be sure to include references to any third-party material you have used.

This document should be written in 12pt font, and the maximum length of the document is 2 sides of A4 (excluding references).

3 Submission

You should submit *three (3) files* for coursework 2. This will be a `pdf` file containing your report, and two `tar.gz` or `zip` files containing your code for the basic game and your extended version of the game in separate archives.

Component (weight)	**Fail [0, 40)**	**3 [40, 50)**	**2.ii [50, 60)**	**2.i [60, 70)**	**1 [70, 100]**
Basic Game (40%)	MIPS code fails to provide the requested functionality. Code may be poorly presented, to the extent that interpreting the code is not possible. Might be no comments in the code or `syscalls` may have been used that are not permitted.	Basic game structure is present. Some functionality may be missing, or the code is too difficult to follow precluding further evaluation.	The game appears to work well, and all the basic functionality is present, but upon further testing bugs start to appear.	The game works well, with well-presented code that adheres to all the expected register and code conventions. The approach taken to some functionality could be improved, but this does not detract from the gameplay.	A professionally-presented piece of MIPS code, adhering to all the expected register and code conventions. Functionality is implemented in an extensible, yet efficient manner.
Game Extension (30%)	Game extension not attempted, or code too poorly presented to be interpretable. Some `syscalls` may have been used that were not permitted.	Game extension attempted but functionality either not completed or not integrated into the main game. Code may not follow register/code conventions appropriately.	Game extension attempted and appears to work without any major bugs. Integration with main game appears reasonable, but there may be some edge cases where the game breaks. Code follows register/code conventions and has reasonable comments throughout.	Game extension has been well-designed and integrated fully into the main game. Testing has been completed to ensure no bugs remain. Code follows all appropriate conventions, but some small improvements could be made to improve efficiency and extensibility.	A professionally-presented game extension, designed and implemented in a creative and engaging way. Functionality is implemented in an extensible, efficient manner.
Report (30%)	No documentation provided, or so poorly presented as to preclude reading.	Documentation is provided, but might not follow a coherent structure, has many spelling and grammar issues that make understanding difficult, or may not follow appropriate academic referencing standards.	Documentation follows a logical, coherent structure. There might be a few spelling/grammar mistakes but these do not impede the reader too much. High-level approach is presented but may not be clear how it relates to the game and/or the code; rationale for each choice may not be presented or is unclear.	Documentation follows a logical, coherent structure. The thought process and decisions are clear to the reader, with appropriate rationale presented when needed. A few minor spelling/grammar issues may persist, but this does not distract from the overall message.	Documentation follows a logical, coherent structure that helps to guide the reader through the development of the game and extension. Rationale for decisions are presented in a clear and consistent manner, backed up by evidence from the literature or preliminary testing where required. Minimal, if any, spelling/ grammar issues.