MODULE NAME:	MODULE CODE:
GAME DEVELOPMENT 2A	GADE6221

ASSESSMENT TYPE: POE (PAPER ONLY)

TOTAL MARK ALLOCATION: 100 MARKS

TOTAL HOURS: A MINIMUM OF 45 HOURS IS RECOMMENDED TO COMPLETE THIS ASSESSMENT

By submitting this assignment, you acknowledge that you have read and understood all the rules as per the terms in the registration contract, in particular the assignment and assessment rules in The IIE Assessment Strategy and Policy (IIE009), the intellectual integrity and plagiarism rules in the Intellectual Integrity Policy (IIE023), as well as any rules and regulations published in the student portal.

INSTRUCTIONS:

1. ***No programming code may be copied from original sources, even if referenced correctly.***
2. ***Make a copy of your assignment before handing it in.***
3. *Assignments must be typed unless otherwise specified.*
4. *All work must be adequately and correctly referenced.*
5. *Follow all instructions on the PoE cover sheet.*
6. *This is **a group assignment.***
7. *You may work in groups of a maximum of **two** members. There are sections where each **individual** member of the group must complete an activity.*

Background

This Portfolio of evidence comprises of two Parts and the final POE where you will submit a fully functional game. Each Part takes you through a step in the game development pre-production process. You will create a fully working endless runner game in Unity in the POE.

This POE has both individual and group elements and is broken down as follows:

- Part 1 – Individual and Group
- Part 2 – Group
- POE (Part 3) – Group

The steps for creating the endless runner are:

1. Development, planning and Game Engine prototype (Part 1)
2. A Single "Level" (Part 1)
3. More Levels (Part 2)
4. Balancing and Final Assets (POE)

Part 1 is similar to the ideation that you will be doing in Game Design 2A. You need to roughly plan out the scope of your game. However, due to this module's focus on game development, most of this ideation will involve development planning in C# and Unity using classes, scripts and other elements as part of your game framework.

This Part will also have you develop a **prototype** that contains the basics of your endless runner – by spawning obstacles, movement, and pick-up mechanics. Colliding with any obstacle causes death. Death terminates that instance of the game. The player should then be able to choose if they would like to exit or replay the game.

Additionally, all endless runners have a **score counter** that you must show on the screen, The scores are affected by pick-ups, objects dodged, etc. The prototype does not have to be a fixed "level" but must include some basic assets. In Part 2, you will import models and elements to create your full aesthetic.

Part 2 has you designing a **single**, complete, designed level of your envisioned game. This level is the first of two levels that must be complete for your final portfolio. In creating this level, you must import assets and develop a full "level" for an endless runner. While the terrain of the game will remain fixed for both levels, the hazards (pits, enemies, obstacles) must be generated randomly in Part 2.

Additionally, halfway through the level, a boss enemy arrives to add an additional hazard type (falling objects, random spikes, collapsing ground, etc.) to make the level a little more difficult. However, even though the game is an endless runner, at some point the player will reach the end of the level (after passing or beating the boss), which is how your levels will change between the two levels for your final POE.

The final part of the **POE** requires that you use this first complete level (Part 2) to build the other level on. Your navigator will give you feedback on the first level which you need to take into consideration to correct the issues and tweak your game. Once the game feels balanced, you will design one more level, with one more distinct "boss" and more assets that you will have created in your Game Design modules or downloaded.

Additionally, you will finalise your art assets, put sound into your game, and string both levels together in an endless loop, where players' scores continuously increase until they die. Finally, upon death, you will allow players to enter their details to place their high score in a database, and you will allow players to view that high score from the game start screen.

Once these phases are complete, you will have a small, two level, looping endless runner game.

Instructions

Each of the Parts build on top of the previous Part. Be sure to complete or update one Part before moving onto the next Part. Each Part will be individually assessed by your lecturer, and feedback for each Part will be provided. Due dates for these individual Parts will be provided by your lecturer.

It is important to note that you will also be judged on the general complexity of your final game.

Though the brief provides you with a degree of implementation freedom, it is still expected that implementation is suitably complex given your knowledge as second year students. Guidance will be given by your navigator but keep this in mind when developing your idea and your game itself.

DO NOT LEAVE WORKING ON THE PARTS AND POE UNTIL THE LAST MINUTE. IT REQUIRES A SUSTAINED EFFORT TO ACHIEVE A GOOD MARK.

IF YOU ARE NOT SURE OF WHAT YOU NEED TO DO OR IF YOU NEED CLARIFICATION OF THE INSTRUCTIONS, PLEASE ASK YOUR NAVIGATOR FOR ASSISTANCE.

PLEASE NOTE: ANY COPYING OF CODE FROM ANOTHER STUDENT, AN ONLINE RESOURCE, A TUORIAL, A TEXBOOK, OR ANY OTHER SOURCE THAT IS NOT YOUR OWN WORK COUNTS AS PLAGIARISM.

PART 1 — DEVELOPMENT PLANNING AND BASIC PROTOTYPE (Marks: 100)

At the end of this specific part, students should be able to:

- Develop digital games using the Unity IDE by including the various tools, menus, editor features and importing assets from other programs.
- Integrate the C# principles to develop games in Unity.
- Access and manipulate GameObjects and Assets in a game.

During this Part, you will **individually** develop an **idea** for your game and write a short report detailing your idea. You will have done this to some degree in Game Design, but this Part requires you to think about your game from a development perspective. These are the restrictions that your endless runner must adhere to by the end of this POE.

Technical requirements

- Use Unity and C#.
- Your game must be developed using Unity's component model.
- Your game must be 3D.
- Your game must use different scenes to switch between menus and game modes.
- Your game must log information to an external database in a meaningful way. This will be elements such as player high scores, a player leader board, player details, etc.
- Your game must import assets of the environment that you have created yourself (from your Game Design modelling knowledge).

Gameplay requirements

- Your game must contain two unique levels with two unique "boss mechanics" – unique things that happen in the level.
- A level's boss "appears" halfway through a level.
- Your game must include at least three different types of pick-ups.
- Pick-ups can be manual (activated via button press) or automatic (activated on pick-up)
- Pick-ups must be time-based (they expire after a period of time).
- You must use Unity prefabs to create different sections of a level (e.g., have holes the player can fall into, hallways that curve or even sections where you need to jump from a piece of ground over an obstacle to another piece of ground.

- You must have at least three prefabs per level (minimum of six different level prefabs across the two levels).
- Each level should randomise between its (at least three) prefabs to create a never-ending course.
- Your player must have the ability to at least move, jump and pick up power-ups.
- Your game will randomly generate obstacles your player needs to dodge.
- Your game must randomly generate the positions of hazards and pick-ups.

Having understood these limitations, create documentation (a Word document: See the word count for each section below) for your game discussing the following:

INDIVIDUAL: PART 1 ACTIVITY 1 – DESCRIPTION	WORD COUNT	MARKS
Rules: All games contain rules. Think about what your pick-ups will do and what your "boss mechanics" will be. Once you have done this, you must define as many of the game's operational rules as possible. This list of rules will directly relate to how you need to program your overall game. Rules can affect either the player or aspects of the environment within the game world. Rules will also impact how your player gains points, moves through the world, uses pick-ups and "completes" each level. Rules also define how the game must interact with your high scores database. Some examples of rules are: • Players can only move left and right, not diagonally. • Players move forward automatically. • Players can jump. • Players enter their name at the end of the game, which saves their high score to the database. Your rules must **discuss** your two boss-mechanics, how three pick-ups work, as well as how your prefab designs will affect the game world. For example: • Pick-up 1 makes the player "fly" by affecting their Y position for a period of five seconds.	500 – 600	20

• Boss mechanic 1 has the boss chase the player from behind. The boss's speed increases over time and can catch the player. • The boss in boss mechanic 1 can fall into the holes in the environment. • The prefabs in Level 1 contain platforms that players must jump between. Your rules should discuss the <u>design</u> of your game in a <u>functional</u> manner (it can be directly translated into code). You should have at least **20 rules**, and it should be clear how the game will be set up just from reading the rules.		
Game Scripts: Once you have defined your game's rules and mechanics, you can start to understand how they will be developed through code. List and describe as many of the C# classes and scripts you will develop and implement in the creation of your game as possible (for Part 1, Part 2 and the final POE), then assign each rule to a specific script. For example: *GameManager.cs* handles the following rules: • Monitors the player's progress passing obstacles to increase their score as they pass an obstacle. • Spawns the obstacles that players need to dodge. • You must describe at least 10 scripts and the rules they will handle.	200 – 400	10
Prefab Types: Your game will contain game objects, such as types of obstacles and types of level terrain. These will be prefabs. You must describe each of these obstacles, include sketches or images of what these obstacles look like, as well as describing the set-up of your level terrain (at least three prefabs per level). You must also describe what scripts will be attached to these objects, if any.	150 – 200	10

Script Relationships: Your scripts will need to interact with each other in various ways. This could be via public *GameObject* variables, a *GameObject.Find()* method call, through inheritance, or another approach. Whatever approach you decide to use, make sure to document these relationships clearly in this section. A diagram is **highly recommended** for doing this.	100 – 200	10
TOTAL	**950 – 1400**	**50**

Once you have completed your individual planning documents **as a group,** take the best parts of each of your individual designs and create a prototype for **one** game that showcases the following:

GROUP: PART 1 ACTIVITY 2 – PROTOTYPE	**MARKS**
The ability for the player to move	5
The implementation of at least one pick-up	10
The implementation of one level's set of (at least three) prefab designs to present a challenge to the player	5
The random spawning of obstacles	5
The destruction of level assets once they are no longer needed in the scene	5
The death of the player when colliding with an obstacle using collisions	5
When a player passes an obstacle, their score is increased by one and should be updated on the UI	5
Display of score on death	5
Ability to restart the prototype	5
TOTAL	**50**

Once you have completed these Parts, upload your planning document, your Unity Project and your Built Unity Game to VegaLearn.

ASSESSMENT SHEET (MARKING RUBRIC)

Please note: Tear off this section and **attach** it to your work when you submit it/ If this is an online submission, then this information needs to be included in the online submission.

MODULE NAME:	**MODULE CODE:**
GAME DEVELOPMENT 2A	**GADE6221**

STUDENT NAME:
STUDENT NUMBER:
GROUP:

PART 1

Marking Criteria	***Fail/Does not meet the required standard (0% – 49%)***	***Average/Meets the required standard (50% – 64%)***	***Above average/Is above the required standard (65% – 74%)***	**Excellent *(75% – 100%)***	**Feedback**
Documentation: Rules	Rules not described.	Only generic rules are covered and need more detail. Unique mechanics are barely present in the ruleset.	Most generic rules are covered, but specifics for unique mechanics are not explained properly.	Most/all the rules, especially pick-up, level and boss mechanic rules are described in detail.	
[20 Marks]	**0 – 9 Marks**	**10 – 13 Marks**	**14 – 16 Marks**	**17 – 20 Marks**	

Documentation: Game Scripts	Game scripts not described.	Very few scripts are identified or described. Not all rules are linked to their respective scripts.	Many game scripts are identified correctly, but rules are often handled by scripts that shouldn't handle them.	Many game scripts are identified correctly and correctly linked to the rules they will handle.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Documentation: Object Types	Game objects not described.	Not all level or object prefabs are described, there are level prefabs missing.	All prefabs are described, but level prefabs are not supplemented with diagrams or sketches.	All prefabs are described, and level prefabs are described in great detail, with sketches, diagrams or other images.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Documentation: Script Relationships	No relationships described.	Relationships described but not correct, or not logical.	Relationships are well described, but only for the base game.	Relationships are well described and include additional features such as the UI and Data-bases.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Prototype: Movement	No movement.	Player can only move forward.	Left and Right Movement does not feel smooth, but forward works.	Movement forward, left and right working.	
[5 Marks]	**0 – 2 Marks**	**3 Marks**	**4 Marks**	**5 Marks**	
Prototype: Pick-up Implementation	No pick-up.	Pick-up implemented, but only works intermittently.	Pick-up implemented, but rudimentary implementation.	Pick-up implemented and reasonably com-plex.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	

Prototype: Prefab Level Design	No unique level (grey-box pathway or similar).	Fewer than three prefabs implemented.	All three prefabs are implemented but they are too simple and present no challenge.	All three prefabs are implemented and present a challenge to the player.	
[5 Marks]	**0 – 2 Marks**	**3 Marks**	**4 Marks**	**5 Marks**	
Prototype: Spawning	No obstacles.	Obstacles only present at the start of the game, no new spawning.	Obstacles spawn but stop after a period.	Obstacles spawn non-stop as the player progresses.	
[5 Marks]	**0 – 2 Marks**	**3 Marks**	**4 Marks**	**5 Marks**	
Prototype: Object destruction	Level assets not destroyed.	N/A	Some, but not all, necessary level assets are destroyed.	All level assets destroyed when they are no longer needed.	
[5 Marks]	**0 – 2 Marks**		**3 – 4 Marks**	**5 Marks**	
Prototype: Player Death and Collisions	Player does not die on collision.	N/A	N/A	Player dies on collision.	
[5 Marks]	**0 – 2 Marks**			**3 – 5 Marks**	
Prototype: Score	Player score does not increase.	Player score increases based on some other metric.	N/A	Player score increases when they pass an obstacle.	
[5 Marks]	**0 – 2 Marks**	**3 Marks**		**4 – 5 Marks**	
Prototype: Score display on death	Score does not display on death.	N/A	N/A	Score displays in a UI element/new screen on death.	
[5 Marks]	**0 – 2 Marks**			**3 – 5 Marks**	

Prototype: Restart **[5 Marks]**	Player cannot restart the prototype. **0 – 2 Marks**	N/A	Player can only restart from death screen. **3 – 4 Marks**	Player can restart the prototype from any point. **5 Marks**	

[TOTAL MARKS: 100]

END OF PART 1

PART 2 — VERTICAL SLICE (Marks: 100)

At the end of this specific part, students should be able to:

- Create and catch custom events within a gaming context.
- Apply modular development to your game.
- Create events with interfaces and delegates.

With your prototype completed in Part 1, you can now use your game engine to design your game's first "level". **It is important for you to incorporate the navigator's feedback you received from Part 1 into your game.**

Despite being an endless runner, your game must consist of two looping "levels" or environments that your player will run through, each with their own unique boss with its own mechanic set. This Part requires you to **complete one of those two levels** by implementing final assets, mechanics and other systems.

In doing so, this level can serve as a vertical slice. This vertical slice is a completed level of your game that is designed to provide players with an example of all the different systems in your game within a single scenario. In a practical sense, a vertical slice of a game is usually a level that a player may encounter towards the middle of a game when they are suitably familiar with mechanics that have been slowly introduced one by one. The game then presents this level to them to test their understanding of this array of mechanics.

To develop your game's first level:

- Complete all the necessary game models that you would need for one complete area of your game (buildings, items, textures, models).
- Implement the remaining two pick-up mechanics and any other mechanics necessary for your game's first level.
- Your level's mechanics and design should be linked to your boss and pick-up mechanics in some way.
- Refine your random generation so that objects do not spawn inside each other or off the map.
- Build your actual level.

Your navigator will have given you feedback on your ideas and implementation in Part 1 about how possible it would be to implement certain mechanics and designs given your knowledge of programming. You may have needed to scale up, scale down or change these ideas based on feedback. However, your final implementation here should still be in line with the level of competency expected by the module. As this is subjective, it is a good idea to continually check in with your navigator about the complexity of your implementation to see whether you're on track. Your game level will be judged on the following criteria:

DESCRIPTION	MARKS
The importing and integration of 3D assets (working animations, custom-built models, etc.)	10
The use of Unity's built-in systems for physics, lighting and other simulated effects	10
Your technical implementation of at least **two** new pick-ups	2 x 5 = 10
Your technical implementation of your level's boss mechanic	5
Your use of Unity's GetComponent() functionality to collect your two new pick-ups, which allows them to be activated	5
Timing the deactivation of your two additional pick-ups	5
Your use of Unity's FixedUpdate() method to control timing for the spawning of your boss mechanic	5
The precision of your level's random generation algorithm: Your objects must not spawn "inside" one another, outside of the prescribed area or too close together to be impossible to pass	10
The presence of UI elements for scorekeeping, pick-ups and other information the player needs to know	10
Your game must be tweaked and balanced so that input, spawning, speed, and other game elements feel responsive ("game feel")	10
The suitable complexity of your implementation of the level terrain, pick-ups and boss mechanic	10
Present and demonstrate your vertical slice to your lecturer during class on a date determined by your lecturer	10
TOTAL	**100**

Once completed, upload your Unity Project folder and built game to VegaLearn.

ASSESSMENT SHEET (MARKING RUBRIC)

Please note: Tear off this section and **attach** it to your work when you submit it/ If this is an online submission, then this information needs to be included in the online submission.

MODULE NAME:	**MODULE CODE:**
GAME DEVELOPMENT 2A	**GADE6221**

STUDENT NAME:
STUDENT NUMBER:

PART 2

Marking Criteria	***Fail/Does not meet the required standard (0% – 49%)***	***Average/Meets the required standard (50% – 64%)***	***Above average/Is above the required standard (65% – 74%)***	**Excellent *(75% – 100%)***	**Feedback**
3D Assets	No additional assets imported (only Unity primitives).	Assets are imported, but very few assets, if any, are custom made, and no animations exist.	Custom-made assets are imported correctly, but animation is badly integrated, or assets could be of higher quality.	Assets are imported properly, have animations where necessary and are of good quality.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Use of Unity Systems	Unity systems not used correctly or poorly implemented.	Only physics systems are implemented and are buggy.	Only physics systems implemented correctly, with no real custom lighting to give mood to the level.	Both lighting and physics systems are implemented to make the game feel “realistic” as far as the word is concerned.	

Marking Criteria	*Fail/Does not meet the required standard (0% – 49%)*	*Average/Meets the required standard (50% – 64%)*	*Above average/Is above the required standard (65% – 74%)*	Excellent *(75% – 100%)*	Feedback
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Pick-up 2 Implementation	No pick-up.	Pick-ups exist in the game but does not activate/deactivate.	Pick-up is working, but the implementation of the pick-up mechanics causes bugs with movement, objects, etc.	Pick-up is working well with no bugs. Activation and deactivation work correctly.	
[5 Marks]	**0 Marks**	**2 – 3 Marks**	**4 Marks**	**5 Marks**	
Pick-up 3 Implementation	No pick-up.	Pick-ups exist in the game but does not activate/deactivate.	Pick-up is working, but the implementation of the pick-up mechanics causes bugs with movement, objects, etc.	Pick-up is working well with no bugs. Activation and deactivation work correctly.	
[5 Marks]	**0 Marks**	**2 – 3 Marks**	**4 Marks**	**5 Marks**	
Boss Mechanic Implementation	No boss mechanic.	Boss mechanic spawns, but has no custom functionality (e.g., just prefabs of obstacles with *RigidBodies* so they fall).	Boss mechanic spawns correctly but does not affect all game elements as it should.	Boss mechanic is working well with no bugs. It spawns correctly and affects the game as described by the rules.	
[5 Marks]	**0 Marks**	**2 – 3 Marks**	**4 Marks**	**5 Marks**	

Marking Criteria	*Fail/Does not meet the required standard (0% – 49%)*	*Average/Meets the required standard (50% – 64%)*	*Above average/Is above the required standard (65% – 74%)*	Excellent *(75% – 100%)*	Feedback
***GetComponent*() to pick up pick-ups**	*GetComponent*() not used.	*GetComponent*() used but pick-up does not activate manually or automatically.	N/A	*GetComponent*() used and allows pick-up to be collected or automatically activated.	
[10 Marks]	**0 Marks**	**2 – 3 Marks**		**4 – 5 Marks**	
Timing the deactivation of two extra pick-ups	Pick-ups do not deactivate.	N/A	Pick-ups deactivate only occasionally.	Pick-ups deactivate correctly.	
[10 Marks]	**0 Marks**		**3 – 4 Marks**	**5 Marks**	
***FixedUpdate*() to control boss spawning**	Boss does not spawn.	N/A	N/A	Boss spawns after a period due to a *FixedUpdate*() based timer.	
[10 Marks]	**0 Marks**			**4 – 5 Marks**	
Precision of random generation	Objects are fixed in the level/prefabs.	Many objects spawn off screen, inside each other, etc. No real effort made to tweak random generation.	Some objects spawn off screen, inside each other, etc., but game balance is still good.	Objects spawn in unique locations with no overlapping, generation contributes positively to balance.	
[10 Marks]	**0 Marks**	**1 – 5 Marks**	**6 – 8 Marks**	**9 – 10 Marks**	

Marking Criteria	*Fail/Does not meet the required standard (0% – 49%)*	*Average/Meets the required standard (50% – 64%)*	*Above average/Is above the required standard (65% – 74%)*	Excellent *(75% – 100%)*	Feedback
"Game Feel" Implementation Tweaks	The game is incomplete.	The game is riddled with bugs or input oddities, making it difficult to assess.	The game feels buggy at times, but overall the experience is smooth.	The game feels smooth to play and work has been put into subtle gameplay tweaks.	
[10 Marks]	**0 Marks**	**1 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
UI elements	No real UI to speak of – just text boxes, etc. No thought behind it.	UI is confusing, incomplete or otherwise difficult to use.	Some obvious UI elements missing.	All necessary UI elements included.	
[10 Marks]	**0 – 2 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Suitable complexity	Overall, the game is too simple and uses little to no custom code (instead using prefabs and Unity systems to "mimic" functionality).	Overall, the implemented mechanics and resulting game need significant work to meet the module standard.	Overall, the implemented mechanics and resulting game meet the module standard.	Overall, the implemented mechanics and resulting game exceed the module standard.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Presentation	Student did not present.	Student doesn't present adequately and focuses on demoing the game.	Student primarily provides insight into their implemented game while presenting.	Student provides unique insight to their game and development process while presenting.	
[10 Marks]	**0 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	

[TOTAL MARKS: 100]

END OF PART 2

POE (PART 3) — FINAL ADDITIONS, DATABASES, INTEGRATION (Marks: 100)

At the end of this POE, students should be able to:

- Connect to a database from Unity.
- Manipulate a database using C#.
- Customise user settings.
- Optimise the game for publishing.

Having developed a full level, there are a few more components your portfolio needs to be complete. During this phase, you will implement these elements. With your game level now complete, you can make any final changes and additions to your game for the final product.

As always, be mindful of any feedback you received in previous Parts from your navigator before finalising your game.

GROUP: DESCRIPTION	MARKS
Level 2: The prefabs that make up your level terrain	10
Level 2: The "boss mechanic" for the level	10
Integrate these two new levels into your game by successfully having the game "loop" between these two levels until the player dies	10
Use your Game Manager to integrate an event system into your game. You must create events for: • Each obstacle being passed (which increments the "obstacles passed" score). • Each separate pick-up being activated (whether automatically or manually). • Each boss "spawning". • A boss being beaten, which now increments a new score component: the number of levels the player has beaten. This will likely require you to do some code refactoring based on your implementation from Part 2. With this event system, your Game Manager should now be tracking these global game events, allowing the game view to update correctly based on these events.	20
User Interface: Using different scenes create: • A main menu which contains the ability to start the game, view player metrics or quit the game; • A game scene with a pause game overlay; and • A game over scene that displays your player metrics and allows the player to go back to the main menu.	5

Sound and Final Assets: Auditory feedback is one of the most essential elements of a video game. As such, you should add appropriate sound to your game. This includes background music, movement sounds, action sounds, sounds made by enemies, and all other sound components that would be relevant to your game type. Add or create sounds and music to your game where appropriate. In addition, import finalised assets to both levels of your game. You will be marked on the integration and completeness of the assets (animations working, etc.)	5
Player Metrics: During ideation, you devised a way to utilise some form of player metrics for your specific game. Implement this metric tracking by connecting your game to a database and storing the relevant information, such as player names and their respective high scores when the players die.	10
Viewing Player Metrics: Implement a way for the player to view these metrics in a separate “high score” menu. The way you implement these metrics and show the result of this metric tracking to users, will depend on your game.	5
Singleton: Edit your Game and Database management objects to implement the Singleton design pattern (so there is only one in any scene). In addition, your Database Manager must be present across both of your game scenes, so it should persist between scenes (not get destroyed when a new scene loads.	5
The suitable complexity of your implementation: Your navigator will have given you feedback on your ideas and implementation throughout about how possible it would be to implement certain mechanics and designs given your knowledge of programming. You may have needed to scale up, scale down or change these ideas based on feedback. However, your final implementation here should still be in line with the level of competency expected by the module. As this is subjective, it is a good idea to continually check in with your navigator about the complexity of your implementation to see whether you’re on track.	10

Presentation: As before, you must present your game to your peers and demonstrate it on this day. While demonstrating the game, you will have to talk through the vertical slice, how you approach its challenges as well as your design decisions for the game.	10
TOTAL	**100**

Once these additions are made, you will have created a fully-fledged endless runner! Submit your Unity project and built Unity game.

ASSESSMENT SHEET (MARKING RUBRIC)

Please note: Tear off this section and **attach** it to your work when you submit it/ If this is an online submission, then this information needs to be included in the online submission.

MODULE NAME:	**MODULE CODE:**
GAME DEVELOPMENT 2A	**GADE6221**

STUDENT NAME:
STUDENT NUMBER:

POE (PART 3)

Marking Criteria	***Fail/Does not meet the required standard (0% – 49%)***	***Average/Meets the required standard (50% – 64%)***	***Above average/Is above the required standard (65% – 74%)***	**Excellent *(75% – 100%)***	**Feedback**
Level 2: Prefab Level Design	No unique level prefabs (basic pathway or similar).	Fewer than three prefabs implemented.	All three prefabs are implemented but they are too simple and present no challenge.	All three prefabs are implemented and present a challenge to the player.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Level 2: Boss Mechanic	No boss mechanic.	Boss mechanic spawns, but has no custom functionality (e.g., just prefabs of obstacles with Rigidbodies so they fall).	Boss mechanic spawns correctly but does not affect all game elements as it should.	Boss mechanic is working well with no bugs. It spawns correctly and affects the game as described by the rules.	
[10 Marks]	**0 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	

Level looping	Levels do not switch or loop at all.	Both levels are played once, and then it sticks to a single level (1, 2 forever).	N/A	Both levels loop, and then randomise after 2 levels (1, 2, 1, 1, 1, 2, 1. etc.).	
[10 Marks]	**0 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Event System	No Event System.	Event System properly integrated but racking less than four unique events.	Event System implemented, but only four to six events implemented and integrated.	Event System properly integrated with game manager listening to all eight required events (pass obstacle, pick-up 1, pick-up 2, pick-up 3, boss 1, boss 2, boss 3, beaten boss).	
[20 Marks]	**0 Marks**	**10 – 13 Marks**	**14 – 16 Marks**	**17 – 20 Marks**	
User Interface	Only a basic game scene exists.	Two scenes exist with some required functionality.	Three different scenes exist but some functionality is missing in some scenes.	Three different scenes exist with all the required functionality in each scene.	
[5 Marks]	**0 – 2 Marks**	**3 Marks**	**4 Marks**	**5 Marks**	
Sound and Final Assets	No sound.	Sound quality is not up to standard (distortion, too soft etc.), many sounds are missing, or models and integration are done very poorly.	Sound is good quality, but models are of average quality and some necessary elements are missing.	Sound and models are good quality, matches the game and all necessary elements/animation exist.	
[5 Marks]	**0 Marks**	**3 Marks**	**4 Marks**	**5 Marks**	

Player Metrics: Database	No database implemented.	Database implemented, but very barebones – only the very basics tracked.	Database implemented correctly, but some useful information left out of metrics.	Database implemented correctly, and all relevant information is tracked.	
[10 Marks]	**0 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	
Viewing Player Metrics	Metrics not viewable in game.	Only a small amount of information is presented to the user.	All relevant information viewable, but presentation to the player could be better laid out.	All relevant information is viewable to the player in an understandable format.	
[5 Marks]	**0 Marks**	**3 Marks**	**4 Marks**	**5 Marks**	
Singleton	No Singleton class implemented.	N/A	Only one Singleton class Implemented.	Multiple Singleton classes implemented (e.g., Both *DatabaseManager()* and EventManager().	
[5 Marks]	**0 Marks**		**4 Marks**	**5 Marks**	
Suitable Complexity	Overall, the game is far too simple and uses little to no custom code (instead using prefabs and Unity systems to "mimic" functionality).	Overall, the implemented mechanics and resulting game need significant work to meet the module standard.	Overall, the implemented mechanics and resulting game meet the module standard.	Overall, the implemented mechanics and resulting game exceed the module standard.	
[10 Marks]	**0 – 4 Marks**	**5 – 6 Marks**	**7 – 8 Marks**	**9 – 10 Marks**	

Presentation **[10 Marks]**	student did not present. **0 Marks**	Student doesn't present adequately and focuses on demo-ing the game. **5 – 6 Marks**	Student primarily provides insight into their implemented game while presenting. **7 – 8 Marks**	Student provides unique insight to their game and development process while presenting. **9 – 10 Marks**	

[TOTAL MARKS: 100]

END OF POE (PART 3)