Assessment 4 – 35%

Complete PacStudent and Design Iteration

Due Dates:

Due via Canvas before **11:59pm Sunday 30th October** (Week 12)

Deliverables:

Unity Project Files:
studentNumber_A4_Project.zip

This is your entire Unity project folder (including your .git folder and .gitignore file), zipped up (as a .zip, no other format), with the specified naming convention. Before creating the zip, delete the "Library" folder.

Student Submission Declaration:
studentNumber_A4_Declaration.txt

A separate submission portal will be open on Canvas to submit a declaration. On this page, you will find a template file to complete and then submit. This template will be similar to the Status_StudentsSubmission.txt files you have completed for each lab activity.

Extensions:

Extensions will only be granted in the case of extenuating circumstances. You must provide evidence of this, such as medical certificates, official letters, photos or video of incident, etc. That you have found the assessment hard or mismanaged you time is not an extenuating circumstance.

Extension requests should be sent to the Subject Coordinator (william.raffe@uts.edu.au). A maximum of 1 week extension can be granted (though the granted time will be proportional to the circumstances. For longer extensions, you must submit an official request for Special Consideration (https://www.uts.edu.au/current-students/managing-your-course/classes-and-assessment/special-circumstances/special-consideration).

Plagiarism Detection Software:

Sophisticated code plagiarism software will be used to check all code submissions in Assessment 3 and 4 with those of other students in the subject, submissions from previous years, and online sources. While learning from online tutorials and communities is strongly encouraged, there is a difference between learning code segments and re-implementing the included concepts or specific Unity API calls, versus plagiarising large chunks of code. A similarity score higher than 25% between a submitted project and any single other source will be further investigated by the Subject Coordinator and, where necessary, raised with the university as Student Misconduct. Additionally, visual and audio assets will be manually checked for originality by marking staff.

You should not be using any assets, packages, or plugins other than those that come with the Unity Installation or that you have created yourself (except the audio assets). This includes any add-on packages found from websites like the Unity Asset Store.

Task and Grade Overview:

In this assessment you are going to be extending upon what you started in Assessment 3 and making the complete PacStudent game. The table below is structured like the lab activities. You should start from the top and work your way downwards. These tasks are stacked in terms of difficulty as well as completing requisite functionality before moving on to later functionality that utilizes this.

You must complete each grade band before moving onto the next. If there are major issues with an earlier section, we will not mark the later ones. E.g. If have made some ghost behaviors but your PacStudent movement is barely working, you will only be marked up to the 70% C level. Likewise, if you skip all of the earlier steps and go straight to the 100% HD Design Iteration, then you will only be marked to the 30% Z level. This is to encourage students to do good, consistent work throughout each section before moving onto the next and stopping student's from thinking they can get easier grades by skipping steps.

This assessment is meant to be challenging.
We do not expect everyone to aim for the 100% HD (or even the 90% HD or 80% D). We follow UTS policy in that a D should be good work, HD should be exceptional work, and a 100% HD is outstanding or near perfect work. Thus, the 100% HD level in this assessment is there to encourage those that want to push themselves. Making games is hard, for everyone. It is a "rockstar" industry that attracts the best-of-the-best software engineers and artist or those with an unwavering passion for games. To succeed in this industry, you always need to be pushing yourself to be better and continue to learn new skills and solve previously unseen problems throughout your life. This subject is no different. We want our students to graduate this degree with as many skills as possible and preparing them for the competitive industry. No matter which degree you are in, push yourself to the highest level you can and be proud of whatever personal achievement that you make for yourself.

Penalties:

The below penalties have been added to discourage certain trends that we have noticed in student work in the past. These penalties can be applied to any level that you get up to, reducing your overall grade:

- Red Compilation or Runtime Errors: If, when opening your project or when playing the game, there are any red errors in the Console Window, you will lose 2 points for each distinct error. This includes compilation errors, IndexOutOfRangeException, NullReferenceException, and others. This does not include Unity Engine errors, which usually come when you first open a project and will disappear if you hit the "Clear" button on the Console Window. We will not seek to break your game, so these errors should not appear in typical gameplay. You may think your game is fine with these errors, but it is not, it is very unsafe to leave them and seeing them in submitted work is a sign of an unprofessional developer. As soon as you see an error (or even a yellow warning) during your development, stop everything you are doing and fix it.
- Resources Folder and Resources.Load(): You will lose 2 points if the Resources folder and/or Resources.Load() is used for anything other than special circumstances in the 100% HD Design Iteration. Some of you have picked up the bad habit of using the Resources folder. This folder is meant for assets that are created at runtime, such as procedurally generated content or player made content, as well as some other rare cases. It should not be used to store sprites, audio files, prefabs, or any other typical game asset. This is because anything in the Resources folder:
    - Must be manually unloaded from memory (e.g. Destroy() and Scene.Load() will not automatically free up the memory)
    - Is kept in its raw form and not compressed with the other game files, making your game larger (a big issue if you are distributing to a mobile store) and making it easier for your assets to be pirated.
    - Causes runtime importing of the assets. When you add an asset to Unity, it is imported to the engine with the settings that can be seen in the Inspector Window. Doing this at runtime will just reduce the framerate of your game.

Overview Videos:

See the video overviews on the Assessment 4 page of Canvas for each grade band to get a better understanding of each. Where ever any discrepancies exist between the videos and this assessment specification document, follow the instructions within this document.

Max grade possible	Requirements

5 (7%)	**Git Repository**
	• You will again be required to use your Git repository in this assessment. You can continue to use your repository from Assessment 3.
	• The marks for the repository will be determined in relation to the band you reach. That is, the number of branches and commits you have should be proportional to the grade band that you reach.
	• You should have a master (main) branch, a dev branch, and at least one feature branch per grade band that you complete.
	  ○ The dev branch should stem from the master/main branch.
	  ○ Each feature branch should stem from dev, unless you have multiple feature branches for a single grade band, in which case subsequent feature branches may stem from the first feature branch in that band.
	  ○ You should make multiple commits per feature branch and merge the feature branches back into dev after the band is completed.
	• Dev should be merged back into master/main before submitting your assessment.
	  ○ Make sure the master/main branch is checked-out before submitting so that your final files are visible to markers.
	• The repository must be connected to a remote online server (e.g. GitHub or BitBucket). You do not need to share this or provide account details, it will be evident from the Git folder whether you have a remote server reference.
	
	Additional resources and suggestions:
	• For more on this style of branching structure, see the Git Workflow branching strategy. You may have more branches if you wish.
	• It is highly recommended that you work on one branch/feature/grade band at a time. E.g. checkout dev -> branch Feature-Audio -> work on the audio while committing frequently -> finish the audio and merge Feature-Audio back into dev -> branch Feature-Visual -> repeat
	  ○ This will help you to avoid merge conflicts. Git helps you recover from poor planning (e.g. resolving merge conflicts), but it is much harder and time consuming than just planning your project flow ahead of time.
	• If something goes wrong - https://dangitgit.com/ (thanks Peregrine7 and Michael on the Discord server for this resource).
	  ○ Git is there to protect from catastrophic failure, as long as you commit and push often.
	  ○ If you mess up a merge, reset back to a previous commit.

	○ If you PC malfunctions, clone the repository from your remote (e.g. GitHub) to another computer. ○ With this in mind, **no extension requests will be granted for reasons of having PC, Unity, project, or Git issues/crashes.**
10.5 (30%, Z)	**Catch-up from Assessment 3** • If you were unable to finish Assessment 3, complete as much as possible in any way possible, up to and including the 75% D band (Manual Level Layout). • The Ghosts should be placed in their center starting area, where they will start the game. • PacStudent should be placed in the top-left hand corner, where the player will start the game • The power pellet flashing animation should be playing. • Remove the movement component for PacStudent from Assessment 3. • Remove the ghost animation cycles (but don't delete the animator from the Project View, you may need it later) • Remove any excess sprites that aren't part of the core level (PacStudent, Ghosts, Walls, Pellets, Power Pellets)
17.5 (50%, P)	**Start Screen UI Appearance and Functionality** • Create a new scene called "StartScene". This will be the first scene of the game and it will be your title screen. • Create a title screen that has at least the following elements (though you can add more if you wish. As always, this should be in your own distinct style, not a replication of the original Pacman title screen): ○ The title "PacStudent" wording ○ A creative subtitle for your game ○ Your "PacStudent" ○ Your "Ghost" sprites ○ An animated border ▪ For example, moving dots around the title or anything else you can think of. ○ "Level 1" and "Level 2" buttons. ▪ When Level 1 is pressed, load up you Assessment 3 scene (that you will build upon in this assessment) ▪ When the Level 2 button is pressed, load the Design Iteration scene (see the 100% HD below). If you do not get to the 100% stage, this button does not need to do anything. ○ Previous high score and time ▪ These can just be 0 and 00:00:00 respectively for now. You will need to adjust them further below when doing high score saving and loading.

	• This scene should be playing the "Game Intro" background music from Assessment 3, on a loop.

21
(60%, P)
	In-game UI Appearance
	• Continue to create your PacStudent re-creation by building upon the scene you were working on in Assessment 3.
	• Add a Screen Space Canvas called "HUD" and add the following elements to the canvas. Aside from the Exit button, nothing else needs to be functional yet, just appear on screen:
	  ○ Lives: A lives counter using the "Life Indicator" sprite from Assessment 3
	  ▪ The player should start the game with 3 lives.
	  ○ Score: A text field to shows the numeric score and that starts at 0
	  ○ Game Timer: A text field that shows a numeric timer in the form of mm:ss:msms (e.g. 01:32:25 is 1 minute, 32 seconds, and 250 milliseconds). This should display 00:00:00 at the start of the game.
	  ○ Ghost Scared Timer: A text field that shows seconds as an integer that will appear and count down when the ghosts are in the scared state. This should be invisible at the start of the game.
	  ○ Exit Button: When pressed, the game should go back to the Start Scene from the previous section.
	• Aspect Ratio: Both the Start Screen menu and this in-game HUD should scale to fit both 16:9 and 4:3 aspect ratios in landscape mode.
	• Add four World Space Canvases called "Ghost1Canvas" through to "Ghost4Canvas":
	  ○ Each ghost should have one of these world space canvas as a child object.
	  ○ Each canvas should have a single text box that has a number from 1 to 4 in it such that your ghosts are labeled 1, 2, 3, and 4 in the game view.

24.5
(70%, C)
	PacStudent and Cherry Movement
	• Create a script called "PacStudentController.cs". In this script, implement PacStudent's movement in the following way:
	  ○ Use a linear lerping/tweening approach to move from one grid position to another. This grid should align with the LevelGenerator.cs levelMap. I.e. PacStudent should be lerping from one pellet position (or empty space) to another at a fixed speed and be frame-rate independent.
	  ○ You may NOT use external tween libraries (such as DoTween). If these libraries are found in your project structure, you will be severely penalized.
	  ○ In Update(), gather player input for moving with the W, A, S, and D key to move PacStudent up, left, down, and right respectively.

- Store the last key that the player pressed in a member variable called "lastInput".
- Do not override/clear this variable's value unless the player provides more input

- In Update(), if PacStudent is not lerping (i.e. was not previously moving or has just reached a grid position), then…
  - Check lastInput and try to move in that direction to the adjacent grid position.
  - If the adjacent grid position from lastInput is walkable, then store lastInput in a member variable called "currentInput" and lerp to the adjacent grid position.
  - If the adjacent grid position from lastInput is not a walkable area (e.g. a wall), then...
    - Check currentInput (see below) and try to move in that direction to the adjacent grid position.
    - If the adjacent grid position from currentInput is walkable, then lerp to the adjacent grid position.
    - If the adjacent grid position from currentInput is not a walkable area (e.g. a wall), then do nothing (PacStudent stops moving).
- With this setup, PacStudent's movement should feel like the reference example except with a slight delay in reaction time if the player provides input while PacStudent is lerping https://www.google.com/search?q=play+pacman+doodle
- See the "Part 4" video overview on the Assessment 4 page of Canvas.uts.edu.au for a clear visual example of how this should function.
  - No, you may not do PacStudent's movement in any other way.
  - No, you may not use Rigidbody physics to handle movements.
  - Yes, we know it isn't the common way of doing things, that is the point. It is unique and we are checking to see if you can program unique systems without following the thousands of unimaginative Unity tutorials out there that lead to the same uninspired feeling in games.

- Make sure that PacStudent's movement animations and audio play when they are lerping, and stop when they are not moving
  - Remember that there are two movement audio clips – one for when PacStudent is eating a pellet (or about to eat one) and one for when they are moving but not about to eat a pellet in the next grid position.
- Create a custom dust particle effect and play it around PacStudent when they are moving so that it looks like they are running through dirt (or a similar surface)

	• Create a script called "CherryController.cs" to implement the spawning and movement of the bonus cherry sprite created in the previous assessment.
	  ○ The bonus cherry should spawn once every 10 seconds.
	  ○ It should be instantiated at a random location just outside of the camera view on any side of the level. The starting location should be different every time.
	  ○ It should move in a straight line, via linear lerping, from one side of the screen to the other, passing through the center point of the level and ignoring collisions with walls, ghosts, and pellets.
	  ○ If the cherry reaches the other side of the level, outside of camera view, destroy it.
	  ○ See below for what to do if PacStudent collides with the cherry.

28
(80%, D)
	Collisions, In-game UI Functionality and Saving High Scores
	• Register collisions between PacStudent and the following elements and have the following effects:
	  ○ As per to original rules in the Assessment 3 specifications, you may only use Rigidbody components to help with registering collisions (e.g. to allow OnTriggerEnter() method calls to between more objects, refer to the Unity collision/trigger matrix on https://docs.unity3d.com/Manual/CollidersOverview.html).
	  ▪ You may NOT use the Rigidbody to handle collision responses (e.g. to put force onto Pacman).
	  ▪ Therefore, any Rigidbody components in your game must be set to IsKinematic = true in the Inspector Window (or Body Type set to Kinematic if you are using a Rigidbody2D).
	  ○ Walls:
	  ▪ Prevent movement in that direction and (if need be) make sure PacStudent is moved back to the previous lerp position before the collision happened.
	  ▪ Create a small particle effect and play it from the point of collision of the first frame of the collision so it looks like PacStudent has bumped into the wall.
	  ▪ Play the wall collision sound effect on the first frame that PacStudent collides with the wall.
	  ○ Teleporters:
	  ▪ When PacStudent gets to the end of one of the empty tunnels on the left and right side of the map, move them to the tunnel on the other side.
	  ▪ After teleporting, PacStudent's movement should continue from that point, moving inwards towards the rest of the level.
	  ○ Pellets: Destroy the pellet and add 10 points to the player's score and update the UI element for this.

- Bonus Cherry: Destroy the cherry and add 100 points to the player's score.
- Power pills:
  - Change the Ghost animator state to "Scared".
  - Change the background music to match this state.
  - Start a timer for 10 seconds. Make the Ghost Timer UI element visible and set it to this timer.
  - With 3 seconds left to go on this timer, change the Ghosts to the Recovering state.
  - After 10 seconds have passed, set the Ghosts back to their Walking states and hide the Ghost Timer UI element.

- Ghosts – Walking State:
  - PacStudent dies and loses a life. Update the UI element for this.
  - Play a particle effect around PacStudent the spot of PacStudent's death.
  - Respawn PacStudent by moving them to the top-left hand corner, where they started the game, and wait for player input.
- Ghosts – Scared and Recovering States:
  - The ghost dies and enters their Dead animator state.
  - Change the background music to match this state.
  - Add 300 points to the player's score.
  - Start a 5 second timer (this does not need to be visible). Once this 5 seconds has passed, transition the ghost back to the Walking state.

- Round Start:
  - At the start of the round, on the HUD canvas, show a big count down of "3", "2", "1", "GO!" (each displayed 1 second apart).
  - During this time, the Game Timer (below) should remain at 0 and the player and the ghosts should not be able to move (see previous and later sections).
  - After "GO!" has been shown for 1 second:
    - Hide this UI element and start the game.
    - Enable player control and ghost movement (if you complete the 90% section below)
    - Start the background music for when ghosts are in their Walking state, which should loop if it finishes.
- Game Timer:
  - Make the timer functional. It should start at 00:00:00 and count upwards after "GO!" disappears from the screen.
- Start Scene High Score and Time:

	○ In the Start Scene, when the game starts, use PlayerPrefs to load the previous high score and associated time and update the UI element for this (see Game Over below for saving the score) • Game Over: ○ When either every pellet is eaten, or PacStudent has no lives left, show "Game Over" text on the HUD canvas. ○ Stop all player and ghost movement and pause the Game Timer ○ Use PlayerPrefs to save the player's current score and current time if the current score is greater than the previous high score or if the score is the same but the time is lower than the previous best time. ○ The "Game Over" text should remain for 3 seconds and then return the player to the Start Scene. If the high score has changed, make sure to update it on this scene.
31.5 (90%, HD)	**Ghost Movement and Artificial Intelligence** • Place all four ghosts in the center area of the map. • Create a new script called "GhostController.cs" to handle ghost movement. • This should work in the exact same way as the player movement, except instead of waiting for player input, at the end of each lerp each ghost should make a decision where to move next based on the following: ○ No Unity Pathfinding: You are NOT allowed to use the Unity Pathfinding tools. ○ No stopping: In all cases below, the ghosts never stop moving. ○ No backstep: In all cases below, ghosts cannot move back to a grid position that they just came from (i.e. they cannot reverse direction) unless there is no other choice. ○ No walking into wall: The ghost should only move to "valid" positions, such as those that PacStudent can walk on (e.g. the corridors). ○ Ghost cannot teleport: In all cases below, ghosts cannot use the teleporters. ○ Ghosts spawning: If a ghost is in the spawn area and they are in a… ▪ Dead state: • Remove the 5 second ghost respawn timer from the 80% section. • When a Dead state ghost are in their spawn area, reset the ghost to Walking, Scared, or Recovering to match the other ghosts. • If no other ghosts are in the Dead state, change the background music to match the new state. ▪ Walking/Scared/Recovering state: Move directly to leave the spawn area out (one) of the gaps in the spawn area walls.

	▪ After exiting the spawn area, ghosts cannot re-enter unless they are in their Dead state. ○ Walking state ghosts: Match the below ghost behaviors with the number above each ghost's head (see in-game HUD – world space section above). ▪ Ghost 1: Move in a random valid direction such that PacStudent will be either further than or equal distance to the ghost (straight-line distance) when compared to the ghost's current position. ▪ Ghost 2: Move in a random valid direction such that PacStudent will be either closer than or equal distance to the ghost (straight-line distance) when compared to the ghost's current position. ▪ Ghost 3: Move in a randomly selected valid direction ▪ Ghost 4: Move clockwise around the map, following the outside wall ○ Scared and Recovering state ghosts: All ghosts use Ghost 1 behavior ○ Dead state ghosts: Move in a straight line, at a constant, frame-rate independent speed, towards the ghost spawn area. ▪ Move through walls and ignore all collisions with PacStudent
35 (100%, HD)	**Design Innovation** This is an opportunity for students to spread their wings and get creative. Once you have completed everything above, create a second scene called "InnovationScene" and link it with the "Level 2" button on the main menu. In this scene, design some meaningful gameplay innovation on top of the existing PacStudent experience. This may include (but is not limited to) ONE of the following: • A weapon / abilities / pickup system that enables extra controls for PacStudent • A small assortment of slightly altered game modes that change the player's objective or rules of the game (see the game "Move or Die" as an example) • A single significantly different game mode that fundamentally changes the PacStudent experience. • Enhanced visuals, particle effects, audio, and camera movement for better immersion. • Procedural generation of the level or other content in the game to increase replayability. • Enhanced ghost AI that behaves more strategically • A full 3D remake of the game with either a 1st or 3rd person camera angle

	• Etc. If you have another idea and you are unsure whether it is too small or too large for the scope of this assessment, talk to your lecturer or tutors about it.