

31263 / 32004 Introduction to Game Development

Assessment 3 and 4

Recreate a Classic Game

Overview:

Assessment 3 and 4 are two parts of a linked assessment. In these assessments you will recreate a specified classic Atari-era game by developing nearly all the assets yourself. The game will look and feel different to the original but will have the same core game design. You will then innovate on the game to push your self-study skills and explore new game development topics on your own. These are **individual assessments**, and it is expected that all the work done here is your own.

Primary Design and Development Constraints:

Regardless of the game that is chosen, there are some important constraints that must be adhered to:

1. You must use the Unity version that is used in the lab room and specified on Canvas.
2. The game must be made in 2D, using sprites.
3. Your final game (in Assessment 4) will have at least three scenes – one that is for the main menu, one that is a recreation of a level from the original game, and the last that shows off your design innovation. You may have more scenes in your project structure if you wish.
4. All visual assets must be entirely your own creation. These must be different in appearance to the original game, and they must be your own designs so as not to infringe on copyright or trademark. i.e. do not draw existing characters from games, anime, movies, memes, etc.
5. You must have sprite animations in your game. You may also create your own particle system effects with 2D sprites.
6. Audio must be included for all major interactions in the game, but the audio clips do NOT need to be your own creation. They must be different to the original game, but you may source these from royalty-free audio websites.
7. All scripts in the game must be your own creation. You may use learning resources such as the Unity manual, forums, Unity Answers, and video tutorials, but you must not use any downloaded scripts from the Unity Asset Store, GitHub, or similar. Sophisticated plagiarism checking software will be used to ensure that your code has not been substantially copied from any single source, whether that is other students or online resources.
8. You may NOT use the Unity Rigidbody physics functionality or the standard Unity CharacterController component (or similar). Therefore, all motion must be coded by you as frame-rate independent continuous movement (see Week 5 lecture). If you have a Rigidbody or Rigidbody2D component on a gameobject for the purposes of collision detection, it must be set to “Is Kinematic” at all times (or Body Type set to Kinematic if you are using a Rigidbody2D).

9. You MAY use the Tilemap functionality in Unity, though it is not required. You MAY NOT use the extra Tilemap Animated Tile functionality as it will cause issues in Assessment 4. For Assessment 3 Band 75% D, you should place any animated tiles manually, regardless of whether you used Tilemaps or not.
10. Assessments 4 stacks on Assessment 3 – anything you miss in Assessment 3 will likely affect your Assessment 4 grade as well.

Assessment 3 – 20%

Pac-man (PacStudent)

Visual Assets, Audio Assets, Level Layout, and Git

Reference Example:

For a reference example of Pac-man, go to Google and type “Pac-man” (<https://www.google.com/search?q=play+pacman+doodle>). On the first page, you will find a playable Google Doodle version of Pac-man that you can play in your web browser. This is the reference that these specifications were built upon (other than the Level Generator). Features found in other versions of Pac-man and Ms. Pac-man are not included here. If a specific feature or design consideration isn’t mentioned, then you can use your own judgment to design it into the game – only what is listed in “Task and Grade Overview” below will be graded.

Due Dates:

Unity Project Files:

Due via Canvas before **11:59pm Friday Week 8**

Deliverables:

Unity Project Files:

studentNumber_Assess3.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.

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. Finally, 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:

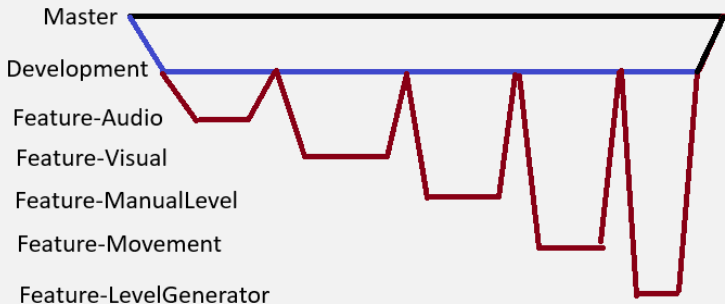
In Assessment 3, you will be assessed on the first half of the semester worth of lecture and lab content by preparing visual and audio assets for your game, setting up and beginning to use your Git repository, and planning for your future development. In this way, you will demonstrate not only foundational knowledge and skills in development, but also your ability to analyse an existing game, decompose it into a set of required assets and coded systems, and identify your own knowledge gaps for future development plans.

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 you don't have any animation (you skipped that section), but you did make the level layout, we will not mark the level layout section. This is to encourage students to do good, consistent work throughout each section before moving onto the next and stopping students from thinking they can get easier grades by skipping steps.

Doing more than what is specified below will not get you additional points for this assessment.

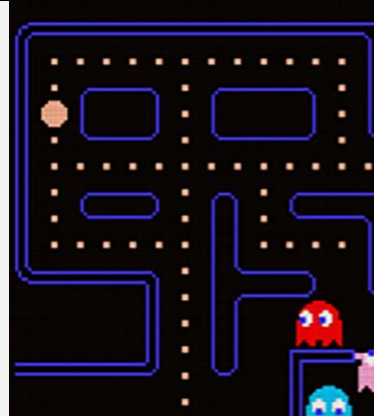
<p><u>Max Grade Possible</u> after completing section and assuming no points lost in earlier sections</p>	<p><u>Requirements</u></p>
<p>2 (10% Z)</p>	<p><u>Git Repo</u> You must set up a Git repository, with a remote host on Bitbucket, GitHub, or GitLab. This git repository must:</p> <ul style="list-style-type: none"> • Have the “.git” folder at the root level of your Unity project folder you should see “.git” next to “Assets”, “Logs”, “Project Settings” etc. in the file structure. • Have the .gitignore file from https://github.com/github/gitignore/blob/master/Unity.gitignore . This should also be at the root level of your project, along with the .git file. • You must commit to your git repository often - e.g. after a few hours of working on the project or every time you reach a minor milestone. <p>By the time you reach 100% HD, your Git repository must have at least seven branches:</p> <ul style="list-style-type: none"> • Master (or Main) - this should be the branch that is active before the submission .zip file is created. • Development – this branch is where the below feature branches are created from and are merged into. This branch should be merged into the Master branch before the submission .zip file is created. • Feature-Audio – this branch should be committed to when adding the audio assets to the project and putting them in the scene. • Feature-Visual – this branch should be committed to when adding the majority of the visual assets and animations into Unity and the main scene. • Feature-ManualLevel – this branch should be committed to during and after completing the manual level layout. • Feature-Movement: This branch should be committed to when you are implementing PacStudent’s movement. • Feature-LevelGenerator – this branch should be committed to during the creation of the LevelGenerator.cs script. <p>As you reach later sections, you will be graded on this section in the following way (all of which is extracted from your .git folder):</p> <ul style="list-style-type: none"> • Having the git repository setup with reference to an online server/remote (e.g. GitHub) • The consistency of commits throughout the project life-cycle. • The use of the above branching structure

	<p>Additional resources and suggestions:</p> <ul style="list-style-type: none"> 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. (See the visual below). <ul style="list-style-type: none"> 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.  <ul style="list-style-type: none"> If something goes wrong - https://dangitgit.com/ <ul style="list-style-type: none"> 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.
<p>4 (20% Z)</p>	<p><u>Project Structure</u></p> <p>The Project Window in Unity should have a logical and organised layout of folders and sub-folders. Folders and assets should be appropriately named such that other potential team members (e.g. your tutors) could easily navigate that project and find the asset they are looking for.</p> <p>The Hierarchy Window of the Recreated Level scene should also be well organized, with the use of parenting to form groups of gameobjects and each gameobject with an appropriate (short but clear) name.</p>

	Inappropriate or messy Project and Hierarchy Window organisation will receive zero or partial marks for this section.
7 (35% Z)	<p><u>Audio Assets</u></p> <p>You must source all of the audio assets that you will need for your game. Like the visual assets, the <u>audio clips used must be different from the original game</u> so as to give your game its own audio style. Do not use assets that are clearly variants of those from the original Pac-man game.</p> <p>You do not need to create the audio yourself. You should only use “Royalty Free” or completely free audio from the internet and not copyright protected audio. If you want to make your own audio, we recommend the use of https://www.bfxr.net/ as an easy online tool to randomize and mix sound effects.</p> <p>Again, you will not be marked for the quality of your audio, but rather the completeness of your audio assets (per the list below) and whether they make sense – e.g. an explosion sound for PacStudent’s movement doesn’t make sense.</p> <p>The audio assets that you will need to include are:</p> <ul style="list-style-type: none"> ● Background music for the game intro (when the level first starts) ● Background music for the “StartScene” scene. ● Background music for when ghosts are in their normal state. ● Background music for when the ghosts are in their scared state. ● Background music for when at least one ghost is in its dead state. ● Sound effect for when PacStudent is moving but not eating a pellet. ● Sound effect for when Pacstudent eats a pellet. ● Sound effects for when PacStudent collides with a wall. ● Sound effect for PacStudent’s death. <p>These audio clips must be imported into Unity and in the Project Window, in a folder named “Audio Clips” and with an appropriate name for each clip.</p> <p>You must also have at least <u>one Audio Source</u> in the scene that, when the Play button is pressed, plays the intro background music and then, when that audio clip either finishes or 3 seconds has elapsed (whichever is earliest), play the background music for the normal state ghosts.</p>

<p>10 (50% P)</p>	<p><u>Visual Assets - Sprites</u></p> <p>You <u>must</u> create all your visual assets yourself. You may use any tool you feel comfortable with. If you do not have much visual design background, it is recommended that you use https://www.piskelapp.com/ or https://github.com/aseprite/aseprite (need to compile yourself or buy precompiled).</p> <p>You must <u>not</u> recreate the assets exactly as they appear in the original game. Instead, you <u>should create your own visual style for the game</u>. These also do not need to be works of art, <u>we are NOT grading you on the quality of your visual assets</u> (we are not an arts faculty). Instead, you will be graded on the completeness of them – you must have a different visual asset and animation for each of the items listed below. The assets should also make sense and be of an appropriate size in your scene (I recommend powers of two) such that it is easy for the player to understand what is going on in the scene. Note that the goal is to recreate the gameplay of Pacman, not the visual style, so make sure to deviate away from the Pacman and Ghosts concept for your art! Some previous works include: chicken & cats, fish & sharks, slime & adventures, and more!</p> <p>The following is a list of visual assets that you will need to create your own version for:</p> <ul style="list-style-type: none"> ● PacStudent in left/right/up/down states ● Pacstudent in dead state ● Four Ghosts in normal (left/right/up/down) states ● Ghosts in scared state (dark blue in original game) ● Ghosts in dead state (just eyes in original game) ● Normal pellet ● Power pellet ● Bonus score cherry ● PacStudent life indicator (bottom left in original game) ● Wall segments for each wall type (see Manual Level Layout section below) <p>All visual assets must be imported into Unity and all visual assets must be placed in the scene view. All sprites that are to be animated must have at least 2 visually distinct frames of animation per animation state (Pacstudent, ghosts (normal, scared, & dead), and power pellet).</p>
--------------------------	--

<p>13 (65% C)</p>	<p><u>Visual Assets – Animations and Animators</u></p> <p>You will also need to create an equivalent Animator Controller (and associated animation clips) for the following animations found in the original game:</p> <ul style="list-style-type: none"> • A single Animator Controller for PacStudent, including the following states and animations: <ul style="list-style-type: none"> ○ Four Walking states, one for each cardinal direction (up, down, left, right). These should each have an appropriate PacStudent movement animation for each direction (mouth opening and closing in the original game). ○ A Dead state and associated animation for when PacStudent dies. • Flashing power pellets • One or more Animator Controllers with the prefix “GhostAnimator_” that do the following: <ul style="list-style-type: none"> ○ A Walking state for moving in each cardinal direction (up, down, left, right) and an associated animation clip for each (eyes look in direction of movement in original game). ○ A Scared state ○ A Recovering state for when the ghosts are transitioning from their scared state back into their normal walking state (flashing between white and blue in original game) ○ A Dead state ○ Hint: If you use one Ghost Animator Controller, use an Animator Override Controller for the other three. Otherwise, use one Animator Controller per ghost. <p>Animations and Animator Controllers must be established so that they can be previewed in the Game View when the Play button is pressed. E.g. PacStudent animator should loop through all movement states and death animation, power pellets should flash in the level, and the ghost animator controllers should cycle through all of it states for each ghost. Each animator state should have an “Exit time” of 3.0 for each state transition or 2 seconds for the whole animation clip (whichever is longer).</p>
<p>15 (75% D)</p>	<p><u>Manual Level Layout</u></p> <p>You should aim to manually recreate the original Pac-man Level 01 layout. When properly constructed, this should look similar to the below image, which is the top-left quadrant of the original Pac-man layout for Level 01.</p>




To recreate this accurately, you should use the map below (represented as a 2D array).

```
int[,] levelMap =
{
    {1,2,2,2,2,2,2,2,2,2,2,2,7},
    {2,5,5,5,5,5,5,5,5,5,5,5,4},
    {2,5,3,4,4,3,5,3,4,4,4,3,5,4},
    {2,6,4,0,0,4,5,4,0,0,0,4,5,4},
    {2,5,3,4,4,3,5,3,4,4,4,3,5,3},
    {2,5,5,5,5,5,5,5,5,5,5,5,5},
    {2,5,3,4,4,3,5,3,3,5,3,4,4,4},
    {2,5,3,4,4,3,5,4,4,5,3,4,4,3},
    {2,5,5,5,5,5,5,4,4,5,5,5,5,4},
    {1,2,2,2,2,1,5,4,3,4,4,3,0,4},
    {0,0,0,0,0,2,5,4,3,4,4,3,0,3},
    {0,0,0,0,0,2,5,4,4,0,0,0,0,0},
    {0,0,0,0,0,2,5,4,4,0,3,4,4,8},
    {2,2,2,2,2,1,5,3,3,0,4,0,0,0},
    {0,0,0,0,0,0,5,0,0,0,4,0,0,0},
};
```

The legend for the above map is shown below. Each of these pieces will require their own sprite image and, as with the other visual assets, should be in your own visual style.

- 0 - Empty (Can either be a background sprite or no sprite)
- 1 - Outside corner (double lined corner piece in original game)
- 2 - Outside wall (double line in original game)
- 3 - Inside corner (single lined corner piece in original game)
- 4 - Inside wall (single line in original game)
- 5 - An empty space with a Standard pellet (see Visual Assets above)
- 6 - An empty space with a Power pellet (see Visual Assets above)
- 7 - A t junction piece for connecting with adjoining regions
- 8 - Ghost exit wall (pink line in original game)

	<p>You must have no more than the 8 sprites above in your Project Window for the level layout. In order to achieve the Level 01 image above, you should place and rotate copies of each sprite to align with other adjacent sprites properly. Some recommendations:</p> <ul style="list-style-type: none"> • Decide a standard size for all sprites (e.g. 16x16 pixels, or 32x32 pixels). • With the above, when imported into Unity, you should be able to leave the scale of all sprite instances as (1,1,1) and they should be appropriately sized. • From here, you only need to deal in 90 degree rotation intervals. E.g. a horizontal wall is turned into a vertical wall by rotating it 90 degrees. Or, a top-left corner piece rotated by 90 degrees becomes a top-right corner, or by 180 degrees it becomes a bottom-right corner piece. <p><u>To produce the whole level</u>, once you have made the top-left quadrant as per the image above, you will then need to duplicate and mirror it horizontally to the right (to create the top-right quadrant), and then both of these are mirrored vertically (to get the bottom-left and bottom-right quadrants).</p> <p>Note that there should be tunnels to the left and the right of the level, one PacStudent in width (to let PacStudent through) but with no pellets in it. Make sure to not double up the on the bottom row after the vertical flip.</p> <p>This level layout should be manually created and present in the Scene View before the Play button is pressed. When the play button is pressed, the camera should show the entire level layout (i.e. all four quadrants).</p> <p>You may use Unity Rule Tiles if you wish (still using only the 7 sprites specified before), but you can also lay the tiles out individually using each sprites Transform in the Inspector panel.</p> <p>Be aware that if the level layout appears identical to the entire original Pacman Level 01 layout; you will receive a 0 for this section.</p>
17 (85% HD)	<p><u>PacStudent Movement</u></p> <p>Use <u>frame-rate independent motion</u> and <u>programmatic tweening</u> from Weeks 6 and 7 to move PacStudent. Do not use methods or parameters like <code>rigidbody.velocity</code>, <code>Vector3.MoveTowards()</code>, etc. to move PacStudent.</p> <ul style="list-style-type: none"> • Move PacStudent clockwise in a cycle around the top-left inner block of the level as shown in the figure below. • The movement must be linear – that is, PacStudent should move at a consistent speed throughout the movement from one corner to the next and this speed needs to be same for every section of the trip.

	<ul style="list-style-type: none"> ● PacStudent must play their movement animation and their moving audio (for when they are not colliding with a pellet) as they moves. ● For Assessment 3, <u>you do NOT need</u> to: <ul style="list-style-type: none"> ○ Respond to user input ○ Detect collisions with walls ○ Detect collisions with pellets. ○ Move the ghosts at all 
<p>20 (100% HD)</p>	<p><u>Procedural Level Generator</u></p> <p>Create a new script called <u>LevelGenerator.cs</u> and copy the levelMap 2D array <u>from earlier into it.</u></p> <p>When the Play button is pressed, the Start() method of this script should delete the existing Level 01 from the scene. From here, LevelGenerator.cs will programmatically procedurally generate the level. To do this, you should take the levelMap 2D array from the “Manual Level Layout” section and the sprites you created for that section and procedurally generate the level layout by stepping through the array and calling Instantiate(...) (and other methods) to create, move, and rotate the sprites. You can also use Tilemaps for this section if you prefer but note that past students have found it challenging.</p> <p>Note that in order to successfully generate the full level you will need to:</p> <ul style="list-style-type: none"> ● Determine the position of each piece ● Determine a way for your code to decide which rotation angle each of the wall and corner pieces should be at to align with each other. Hint: have you code look at the pieces around it to determine how it should be rotated to connect to them (in the case of walls) ● Mirror the above 2D array or instantiated pieces three times (horizontally, vertically, and horizontally-and-vertically) to get the other three quadrants of the level to make the full level.

- For the vertical mirroring, you will need to ignore or delete the bottom row of the 2D array so that there is only a single row of empty spots.
- Remember that a 180-degree flip is not the same as two 90-degree rotations for asymmetrical tiles.
- Adjust the game camera to react to the size of the level such that in Play mode the entire level layout can be seen.

How this will be graded:

When marking this section, your tutors will substitute the levelMap 2D array with a custom one, of any size, using the same legend and a similar style, but a different layout. Your code must adapt to this custom levelMap 2D array to produce a logical level, with marks being removed based on the number of pieces in error (e.g. in the wrong place or with the wrong rotation).

Some other notes:

- This must be done through your own code and you should not use Rule Tiles or any other in-built Unity systems for procedurally generating levels.
- The top-left corner of the level (i.e. levelMap[0,0]) will always be an outside corner piece with a default rotation (i.e. it will look the exact same as the manually built level).
- The walls will always have a smooth connection. I.e. a vertical wall will never be adjacent to a horizontal wall, there needs to be a corner piece or T piece joining them.
- The PacStudent Movement from the 90% band does not need to be updated. Leave PacStudent moving clockwise around the area where the manually create map was.
- Do NOT permanently remove the manual level from your scene even if you complete this section. It is required for you to receive grades for the Manual Level layout grade even if you successfully complete this section.
- Try testing with your own levelMap 2D arrays to evaluate edge-case scenarios as the original csv file does not cover all potential tile combinations and layouts.
- There will not be a tutorial on this or resources provided. Procedural content generation is all about working through logic problems, with an eye on the visual design quality of the output. So, this is a test of your logical thinking skills – get out a pen and paper (or drawing software) and start planning it out before you start programming. If you ever want to do anything interesting in your IT career (especially making games), these logical thinking skills are a must as you encounter new problems. This is also for the 100% HD, which at UTS is often reserved

	for students that “exceed expectations” and go well above the average skill set. So challenge yourself and give it a go!
--	--

End of Assessment 3 Document