Interviewer template

Patch 2b – Week 4 – CMP424

Dr William Kavanagh & Dr Naman Merchant

This submission consists of 15% of your overall module grade

TODO: Fill in all sections marked in **[red]**

Interview Questions for Picture Particle

Either select one of the example jobs from week 1 (available on MLS) or a games job online with clear expectations, including a link to the listing.

Job Role: Junior Generalist Programmer
Student Name: Callum Myers

Question One:

How would you go about planning and implementing a maintainable player character controller in Unity?

Key Skills Examined:

Class structure, modularity, efficiency, implementation

Example Answer:
I would first assess the mechanics required to make the character controller, which would include input, movement, camera, world interaction and physics, as well as any abilities or mechanics specific to the game. I would then create scripts to handle each of these separately, since this would help improve simplicity if any expansions or changes were required for a specific aspect of the controller in future. These scripts would use modular systems such as the Unity Input System, which allows a singular variable to refer to multiple input sources, for example using if ("Jump" == true) could check for a touchscreen tap, keyboard key or controller button and have the same result for all. This would be especially useful if the game were to be released on multiple platforms later down the line since the basic input handling system would not have to be fully revamped to allow for other inputs.

I would use a state machine which constantly checks for a player's action at a given time, for example if they are running, jumping, idle, on the ground or midair, etc. and this would be used to decide if the player's desired next action is possible based on their current state. This would, for example, prevent the player from jumping infinitely while midair, or blocking whilst shooting. This can be especially useful when working with animations, since each animation can be linked to a specific state. It would also improve performance as not all checks would be required in every state, since any checks relating to entering the current state or states which cannot be accessed from the current one could be ignored.

Finally, I would iterate over my created controller once mechanics have been implemented and assess where performance could be improved. This could mean minimizing the number of times physics calculations occur as these can be computationally expensive, reusing objects or animations where possible to minimize the amount being loaded into the game or scene, and ensuring any components which are not required are disabled or unloaded, such as having a player's gun be a separate component and disabled when not being shot.

Prompts and Guidance:

- How could you make sure it would be simple to change specific areas of the code in future?
- Unity has some built in systems which can be very useful when building modular code. Where and how could you use these?
- Give some examples of assets which could be reused, and why doing so may improve performance.
- Ask about what system(s) could be used to decide which actions the player should be able to perform next.

Question Two:

Explain how you would rotate a complex shape by 90° around the Y-axis using matrix multiplication.

Key Skills Examined:

Matrices, matrix multiplication, mathematics

Example Answer:
I would first get the coordinates of each vertex on the shape and convert these into a matrix using [x, y, z, 1]. Next, I would get the appropriate general rotation matrix and using this I would multiply each column of the rotation matrix by the position matrix, which would give me the new position matrix of the rotated vertex. Finally, I would repeat this step for every vertex on the shape before plotting these new rotated matrices which will give me the rotated shape.

Prompts and Guidance:

- What matrices do you need, and which do you currently have?
- Give an example position.
- How many times will you need to repeat this process? (Assuming candidate did not mention repeating for every vertex).

Question Three:

What are 3 potential options for creating AI controlled enemies, and what are some strengths and weaknesses for each?

Key Skills Examined:

AI, understanding, problem solving

Example Answer:

- Finite state machines can be good for being easy to manage as their actions and responses are largely controlled by the developer which can make them consistent and predictable, useful for creating specific outcomes, however this can also be a downside as it can make it too simple for players to overcome it due to being able to guess it's next move.
- Genetic algorithms can change and adapt based on the player's actions and how they play through the game, which can make this model great for dynamic difficulty since the game will gradually get harder as the AI learns to read the player's moves, however it can have very high computational costs if not correctly restricted and managed.
- Fuzzy logic can create outcomes which seem almost like a human is controlling the enemy due to its unpredictability, however it can be more difficult to implement and will require extensive testing to find small errors or edge cases which can break the decision-making process of the AI.

Prompts and Guidance:

- Which models or algorithms are often used to cause a game to get more difficult over time?

- Try thinking about how a player may improve at a game, and consider what AI uses a similar approach to improve AI-controlled characters' decisions.

Question Four:

How do you ensure UI elements are responsive and intuitive for the player?

Key Skills Examined:

UI, problem solving

Example Answer:

- It is important that the UI looks and feels consistent across many screen sizes and resolutions, which can be achieved using appropriate scaling and anchoring of elements.
- Input should feel snappy and have clear feedback that it has registered, which can often be done using a combination of visual effects, such as different graphics for buttons when idle, hovered or pressed, as well as audible feedback, for example a click when a button is pressed.
- There should be as little on the screen as possible while still having the functionality required, as a cluttered or overly complicated UI can make it difficult to find what the player is looking for, or to focus on the level if it is an in-game HUD.
- Similarly functional items on the UI should be kept close together, such as having all volume functions on one section and custom keybinds in a different section, and vital items should stand out more with effective use of size, colour and fonts.
- The UI should constantly be changing and iterating over the game's lifespan as new modes or items are added in, and through consistent testing and user feedback it should be adapting to the thoughts of the end user to make it as convenient and easy to use as possible.

Prompts and Guidance:

- How should the UI look on different screens or platforms?
- How does a player know the UI is working correctly?
- How much should be on the screen at once?
- Try to bring up the idea of user testing and going back to iterate if required.

Question Five:

What are some ways that solid communication between programmers and other departments within the team can benefit the project?

Key Skills Examined:

Communication, collaboration, teamwork, development workflow

Example Answer:

One key benefit is that if the communication is strong, especially early in the project, this can help specific departments keep their goals aligned with the skills of the team. Designers may often come up with strong mechanic ideas which would benefit the game, however the programmers may be able to say early on that it would be too complicated for the time limit and can have it simplified or cut from the project to keep expectations realistic.
It can also lead to creative solutions to problems which could otherwise go out of scope, such as if a game designer wanted to create a vast open-world level but the level designers already have a heavy workload, the programmers could suggest a procedurally generated solution, creating less work for level designers.

Prompts and Guidance:

- Try to have candidate talk about potential links between specific departments.
- Which departments will be active throughout the whole development, and which may work more or less at certain stages?

Question Six:

Explain the steps required to fire bullets from a gun in Unreal Engine 5. Assume there is already a working player controller with a controllable gun.

Key Skills Examined:

UE5, blueprints, C++, scripting

Example Answer:

First, I would create a new Blueprint actor class which will be used as the base of the bullet objects. I would initialize this bullet with some form of model to display the bullet on screen, then add Unreal's projectile movement component, setting an initial and maximum speed. To complete the Blueprint class I would add a collision detection component and set up some simple collision events in the event graph which will cause the bullet to perform actions depending on the collider it hits.
To add the ability to create these bullets I would first add a new action to the input of the project settings and assign an input to it. Next, in the player controller I would add a new node which runs when this input is detected, and in this I would use the Spawn Actor from Class node then set the class to the bullets Blueprint class, which will create an instance of this class each time the event fires. Finally, I would add polish effects such as audio using whichever audio system the project is using, and test to make sure the bullet fires as intended.

Prompts and Guidance:

- What initial variables would need to be stored in the bullet, which would be consistent for every instance? (Model and initial/max speed).
- What variables would need to be passed to the bullet when created? (Spawn position and direction).
- The candidate could also discuss how they would manage ammo counts, reloading times or weapon switching.

Question Seven:

How would you implement a ball which can bounce off the floor but will stick in position after hitting the wall? This should be done using Unity.

Key Skills Examined:

Physics, Unity, implementation, code planning

Example Answer:

First, I would create an empty GameObject and add a sphere component to it to display the ball, as well as a RigidBody component with gravity enabled, and a sphere collider to handle collisions. Next, I would create a new Physics Material which has its Bounciness value set appropriately over 0 and attach this also to the GameObject.

I would then write a script which calls a reference to the GameObject's RigidBody in the Start() function and store this as a variable. In an OnCollisionEnter() function, I would use the CompareTag() function to check the tag of the GameObject connected to the collider which the ball is colliding with, and if this tag returns "Wall" I would set the velocity of the ball to 0 by setting the RigidBody.velocity to Vector3.zero. If it does not return "Wall" I would do nothing since the physics material will cause it to bounce. Finally, I would set the tag component of any walls in the scene to "Wall" and test the system, which should be working.

Prompts and Guidance:

- What subsystems does Unity have which could handle the bounce for you?
- How would you know if you are colliding with a wall or floor? What should you do in each of these scenarios?
- What component of the ball could you use to change the movement of the ball?