S2D: Pacman

Week 3 – Simple Menus

Task 1 – Some improvements to the code

It's always important to consider your coding style. With the framework you were given last week, there were some decisions regarding the coding style which could certainly have been a better.

Constant Data

Firstly, when we have information such as Pacman's Speed it is a good idea to create a variable to store this in. It makes our code easier to read and maintain.

Navigate to 'Pacman.h' and in the 'private:' section of the Pacman class add a constant variable for Pacman's speed using the following code:

```
//Constant data for Game Variables
const float _cPacmanSpeed;
```

The next thing we need to do is set this to be a value, this value can only be set once and will never change. Navigate to 'Pacman.cpp'. You will then need to add a member initialiser to the Pacman constructor method. You can do this by replacing the following code on line 5 from:

```
Pacman::Pacman(int argc, char* argv[]) : Game(argc, argv)
```

To

```
Pacman::Pacman(int argc, char* argv[]) : Game(argc, argv), _cPacmanSpeed(0.1f)
```

This sets the variable `_cPacmanSpeed` to 0.1f.

In the Update method of Pacman where we use 0.1f to change Pacman's X and Y position, we should change it to use `_cPacmanSpeed` instead. For example, we would change the right direction movement to:

```
if (keyboardState->IsKeyDown(Input::Keys::D))
            _pacmanPosition->X += _cPacmanSpeed * elapsedTime;
```

Avoiding hard coded values

In various places throughout the application, the values for width and height were hardcoded as 1024 and 768 respectively. This is never a good idea, what happens if the width and height changes?

In places where you can see a hard-coded value try to replace it. In your collision detection with the edges of the screen, your code may look like:

```
if (_pacmanPosition->X > 1024)
```

You can change this to be much clearer and flexible by altering the hard-coded values

```
if (_pacmanPosition->X > Graphics::GetViewportWidth())
```

Task 2 – Creating a pause menu

Adding and Initialising values

Using the Pacman Framework from the previous tutorial, we will look into what is required to create a simple Pause menu. Firstly we need some data variables to store the information for our pause menu. Navigate to 'Pacman.h' and in the 'private:' section of the Pacman class add the following variables:

```
// Data for Menu
Texture2D* _menuBackground;
Rect* _menuRectangle;
Vector2* _menuStringPosition;
bool _paused;
```

The first thing we need to do is to initialise these variables. We set generally set default values in the Constructor and we load any other resources in the LoadContent() method. The Paused Boolean value requires a default so in the Constructor for Pacman located in 'Pacman.cpp' adjust the section of code shown below to match:

```
Pacman::Pacman(int argc, char* argv[]) : Game(argc, argv), _cPacmanSpeed(0.1f)
{
	_frameCount = 0;
	_paused = false;

	//Initialise important Game aspects
	Graphics::Initialise(argc, argv, this, 1024, 768, false, 25, 25,
"Pacman", 60);
	Input::Initialise();

	// Start the Game Loop - This calls Update and Draw in game loop
	Graphics::StartGameLoop();
}
```

Then in the LoadContent() method, we need to load in the rest of our data. Add the following code to the LoadContent() section:

```
	// Set Menu Paramters
	_menuBackground = new Texture2D();
	_menuBackground->Load("Textures/Transparency.png", false);
	_menuRectangle = new Rect(0.0f, 0.0f, Graphics::GetViewportWidth(),
Graphics::GetViewportHeight());
	_menuStringPosition = new Vector2(Graphics::GetViewportWidth() / 2.0f,
Graphics::GetViewportHeight() / 2.0f);
```

Carefully look at what each of these variables has been initialised to; there use should become a little clearer soon.

You may have noticed that we have loaded a new texture called 'Transparency.png' but we haven't added this to our Textures folder. Go to Blackboard and download 'Transparency.png' and then copy this into the' Textures' folder within the Pacman folders.

It's a good idea to build your solution at this stage to make sure you have no errors. Go to the Build menu and select 'Build Solution'. If everything is ok continue on, if you have errors, take a careful look through the code and try to spot what is causing them, ask your tutor if you can't find what's wrong!

Drawing our pause screen

Now we need to add some code to draw the pause screen. Navigate to the `Pacman::Draw(int elapsedTime)` section in 'Pacman.cpp'. Towards the end of this method, we need to add some extra code to draw the menu. Make sure you enter this code just before the line which says:

```
SpriteBatch::EndDraw(); // Ends Drawing
```

The order we input our code can make a big different especially when we are drawing. Our drawing code will draw in order so the first thing we draw will be drawn first to screen and subsequent draw calls will be layered on top. This is why we want to draw our pause screen at the end. To draw our pause screen, add the following code:

```
if (_paused)
{
	std::stringstream menuStream;
	menuStream << "PAUSED!";

	SpriteBatch::Draw(_menuBackground, _menuRectangle, nullptr);
	SpriteBatch::DrawString(menuStream.str().c_str(), _menuStringPosition
Color::Red);
}
```

This code only executes when the _paused variable is true. It then draws the slightly transparent texture and then a string on top of this.

You can test to see if this has worked by changing the default value for _paused from false to true in your constructor that you set earlier. Try this now and then run your game and see if it draws correctly.

Once you have tested this, change _paused back to false and then we will add some logic to turn the pause menu on and off.

Turning our Pause Screen on and off

In the Update() method, after the set up the keyboardState we need to add some code to pause the game if the P key has been pressed. Add the following code just after you check the keyboardState.

```
if (keyboardState->IsKeyDown(Input::Keys::P))
{
	_paused = !_paused;
}
```

The ! operator inverts a value, so it makes a true Boolean false and would make a false Boolean true. We are using it here to flip the _paused variable between true and false each time the P key is pushed.

The next thing we have to do is wrap the rest of our update code in a big if statement so that our game stops updating itself when it is paused. So, after the code you have just entered but before you check for any further input or collisions for Pacman add the following statement:

```
if (!_paused)
{
```

At the end of your update method, close the brackets to complete this if statement.

This should look something like the following (although with much more code in the middle of the if statement!)

```
if (!_paused)
{
	// Checks if D key is pressed
	if (keyboardState->IsKeyDown(Input::Keys::D))
		_pacmanPosition->X += _cPacmanSpeed * elapsedTime;

	// Checks if Pacman is trying to disappear
	if (_pacmanPosition->X > Graphics::GetViewportWidth())
	{
		// Pacman hit right wall - reset his position
		_pacmanPosition->X = -_pacmanSourceRect->Width;
	}
}
```

You can now try to run this and see if it works! See if you can spot the problems.

Stopping the munchies from animating behind the pause screen

In the draw() method is a line of code which increments the frameCount variable. Take the line of code which looks like:

```
_frameCount++;
```

And move it into the Update() method, within the if(_paused) section you created earlier. Run this and your munchies should now stop animating whilst paused.

Stop the Pause screen flicking on and off really quickly

The reason the pause screen flicks on and off really quickly when you push P if because it checks if P is being pushed down every frame (60 times a second!). We need to change this so that only check new presses of the P key. To do this, add a new variable to 'Pacman.h' like;

```
bool _pKeyDown;
```

Set this value to false in the constructor just like you did previously with the _paused variable:

```
_pKeyDown = false;
```

Then we need to change the P key check to now use this value to prevent it checking all the time and only check again once the P key has been lifted back up. So alter your code which checks if P is pushing in the Update() method to look like the following:

```
if (keyboardState->IsKeyDown(Input::Keys::P) && !_pKeyDown)
{
	_pKeyDown = true;
	_paused = !_paused;
}

if (keyboardState->IsKeyUp(Input::Keys::P))
	_pKeyDown = false;
```

Run this now and hopefully you should have a fully working Pause screen!

Task 3 – Start Screen

As an extra task, try to take the code we used for the pause screen and make a start screen that expects you to press Space to start the game. This screen should be visible before the game begins. You can use very similar logic to the pause screen but there will be a few differences!

Task 4 – Tutorial Plus

Self-managed task

This is an opportunity to catch up on previous tutorials that require finishing or advance your skills using the lecture material from earlier this week. The challenge is to come up with some game logic for Pacman that makes use of the data types sequences.

- Find an appropriate way to put a for loop in your game
- Come up with a way to add some Booleans to your game
- Be creative with your art assets

Ask your tutor for some ideas if you are struggling.

Happy coding!