Title: Fennex

Author: Maddisen Topaz

Dates: 21st March 2017 - 11th April 2017

Purpose:

Fennex is a simple 2D platformer game.

Files:

Animation

- Animation.cs
- AnimationHandler.cs
- TextureReel.cs

Collisions

- BoundingBox.cs

Level Assets

- BackgroundImage.cs
- Crystal.cs

- Platform.cs
- MovingPlatform.cs
- Spikes.cs

Levels

- Level.cs
- DarkForest.cs

Managers

- InputHandler.cs

Particles

- Particle.cs
- ParticleEngine.cs
- SurfaceParticleEngine.cs

Game

- Camera.cs
- Game1.cs
- Player.cs
- Texture.cs

Requirements/Specifications:

Game Description

Fennex is a simple 2D platformer game. You play as Fennex, the fennec fox; your mission is to collect all of the crystals in the level, while avoiding obstacles such as spikes that will cause Fennex to die. Fennex is a light-hearted adventure platformer, aimed at a target audience of ages five and above.

Requirements

- Currently Fennex is not cross platform and can only be played on a 32-bit or 64-bit windows machine.
- The machine must have a DirectX 9.0 (or above) capable GPU.

Monogame Components Used

- Game
- GameTime
- GraphicsDevice
- ContentManager
- GraphicsDeviceManager
- SpriteBatch
- SpriteEffects
- Texture2D
- SpriteFont

- Song
- MediaPlayer
- Rectangle
- Vector2
- KeyboardState
- MouseState
- GamePadState
- Keyboard
- GamePad
- Keys
- ButtonState
- Color

User Guide:

Playing Fennex

Fennex can be played on any 32-bit or 64-bit windows machine. The executable (Fennex.exe) file must be run from the folder with the necessary .dll files and the content folder to play the game.

To play Fennex, simply open the Fennex directory folder and double click on Fennex.exe. To start the level press spacebar.

You can move Fennex around in the level by pressing A for left and D for right. To make Fennex jump press space bar.

Controls

- A- Left
- D- Right
- Spacebar- Jump

Goals

- Avoid the spikes they will cause Fennex to return to the last checkpoint
- Collect all of the crystals and reach the end of the level

Building and Running Fennex

Fennex was written using the Monogame API Version 3.6 and Visual Studio Community 2015. To build and run Fennex you will need to download and install Visual Studio 2015 and then install Monogame 3.6 or higher, for Visual Studio.

You will then be able to build and run Fennex in 32-bit Debug or Release mode

from Visual Studio.

To install Visual Studio Community 2015 see:

- https://msdn.microsoft.com/en-us/library/e2h7fzkw.aspx

To install Monogame for Visual Studio Community 2015 see:

- http://www.xnahub.com/how-to-install-monogame-on-visual-studio-2015/

Structure/Design:

Animation

To play an animation I created an AnimationHandler class which controls the playback of an animation. Each animation must have its own AnimationHandler. When creating a new animation handler you pass in the animation that the handler is in charge of.

The handler is in charge of playing the animation which includes calculating what frame in the animation we are up to based on the current game time. The animation handler can also return the total time the animation should play for, over one cycle.

The Animation class contains all of the information about the animation itself, such as the amount of time to display each frame in the animation, whether the animation is looping and the texture data (frames) of the animation.

I adapted the design of the animation classes from a simple monogame platformer I found online.

See the following link:

https://github.com/MonoGame/MonoGame.Samples

The original classes (Animation.cs and AnimationHandler.cs) were adapted from the Platformer2D project. They were designed to cycle through a sprite with a single texture that contained all of the frames of the animation. The width of a frame was assumed to be same as the height of the texture.

For example:

I needed to be able to support sprites with multiple textures, as my sprites such as the run animation, had frames that were separate image files. I also had frames where the texture was not square.

Thus I modified the Animation.cs class to store a TextureReel object, which consists of a list of textures, where each texture is a frame in the animation. I can then add the required frames for the animation to the texture reel one by one. I also have the option to pass a spritesheet into the animation, where a texture reel will be constructed (the texture is split into frames).

Collisions

The player collisions makes use of two classes; BoundingBox.cs and Texture.cs.

The Texture class is wrapper to the XNA Texture2D component. Creating a Texture object allows me to use all of the Texture2D functionality, with an additional function GetPixel(). This allows me to access the pixel data of a Texture object.

To check for collisions between the player and the objects in the world I do the following:

1st Check - Bounding Box Collision

- Iterate through each collidable object in the world and generate its bounding box
- Generate a bounding box for the player and check if the two bounding boxes are colliding

2nd Check -Pixel Collision

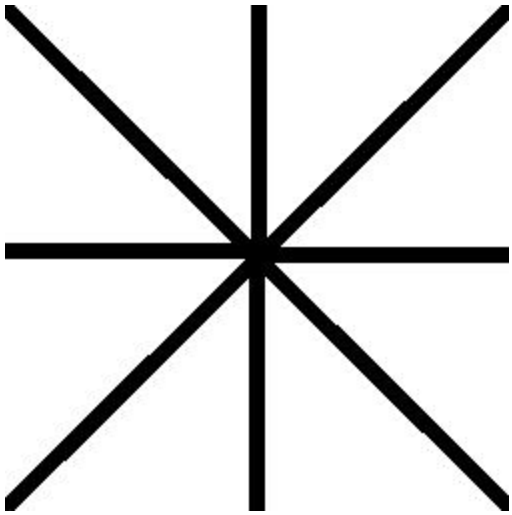
- If a bounding box collision has occurred then check if a pixel collisions has occurred
- First generate two bounding boxes; one for the player and one for the platform
- Next get the bounding box formed by the intersecting region of the two boxes
- Iterate through the pixels within the intersecting region from the players texture
- If the pixels alpha is greater than 0 then transform the pixels coordinates into the image space of the second texture, where the second texture is

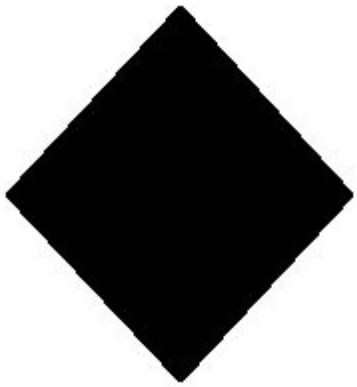
the texture of the object the player is colliding with
- If that pixels alpha is greater than 0 then a collision has occurred

Once a collision is determined to have occurred the next step involves fixing the position of the player so that a collision is no longer occurring.

I used a simple island search to find a place to move the player back to in order to stop the player colliding.

The island search is designed to search for a new position to place the player in such that it is no longer colliding with the objects in the world.





The island search iterates in a star shaped pattern, by first trying to move the player up, then down, then left, then right, then diagonally in each direction; by one pixel, then two and so on, until a suitable position is found where the player is no longer colliding with the objects in the world.

This method was an optimisation on the diamond shape island search, which was substantially slower.

<u>Player</u>

The player class is essentially a simple state machine. The current state of the player determines which animation should be playing and what actions are allowed while the player is in that state.

To get the new state of the player I use a 2D array which takes two parameters, the current state of the player and the input action. This array takes the two values and gives back a third value, the new state the player should be in.

The current state of the player is used as the index to an array of animation handlers which determines which animation to draw the player in.

Input

The InputHandler.cs class is used to handle input in the game.

I designed the input handler to keep track of the keyboard state, mouse state and also gamepad state. This allows me to query the input handler to check if any key is down or pressed, any button on the gamepad is down or pressed, or if the left or right mouse button is down or pressed.

Particles

I adapted my Particle.cs and ParticleEngine.cs classes from the following tutorial:

http://rbwhitaker.wikidot.com/2d-particle-engine-1

I followed this tutorial fairly closely as it already did exactly what I needed it to do.

However I did make changes to the design of the update function. In the original tutorial particles are added every update, which causes the particle generator to generate many particles each frame. I wanted to be able to control the flow of particles from the particle generator as some things may emit more or less particles per second.

To implement this feature I added a new parameter particles per second, which checks how many particles the update function needs to create based on how much time has elapsed and how many particles have already been created.

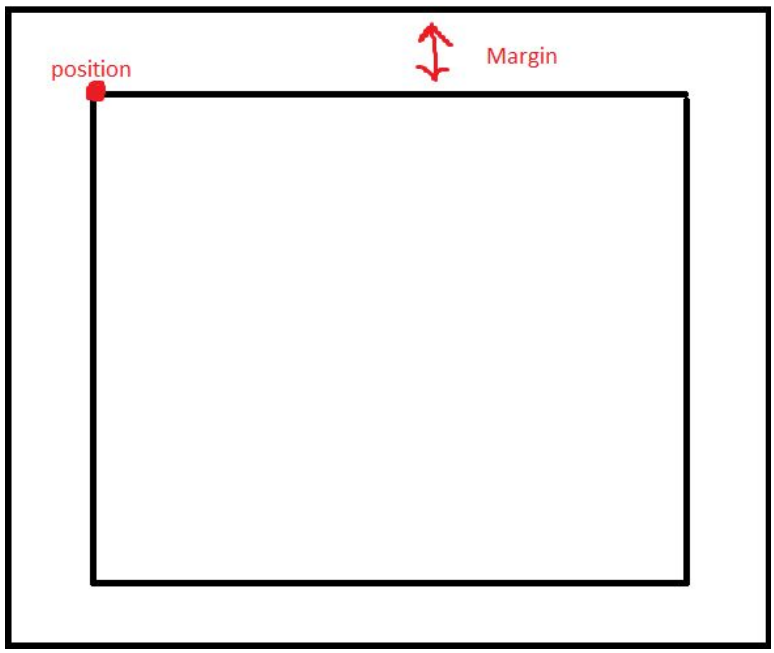
I also wanted the particles to fade out over time, rather than simply be removed as they were in the original tutorial. To do this, I changed the particles draw function, so that the color is multiplied by an alpha scale, which is calculated based on how much time to live the particles has left.

Furthermore, I also expanded on ParticleEngine.cs by adding SurfaceParticleEngine.cs, which is a particle emitter that spans over an entire surface. The SurfaceParticleEngine also only adds an upward velocity to the particles, which makes the surface appear to emit particles from only one side of the surface.

It was my plan to implement particles stages as well, where each particle would cycle through a list of its own textures that were based on the life of the

particle. This would have added even more realism to the particles and allowed me to do more complex particle emitters such as fire. However I wasn't able to implement this change in the time frame.

Camera



The Camera class contains the position of the top left corner and the margin between each edge of the camera's bounding box and the screen size.

To move the camera with the player I check if the player is standing within the margin of the camera, I then move the camera so that the player is inside the camera's viewport again. This is done by drawing all images in the world at their position minus the position of the camera.

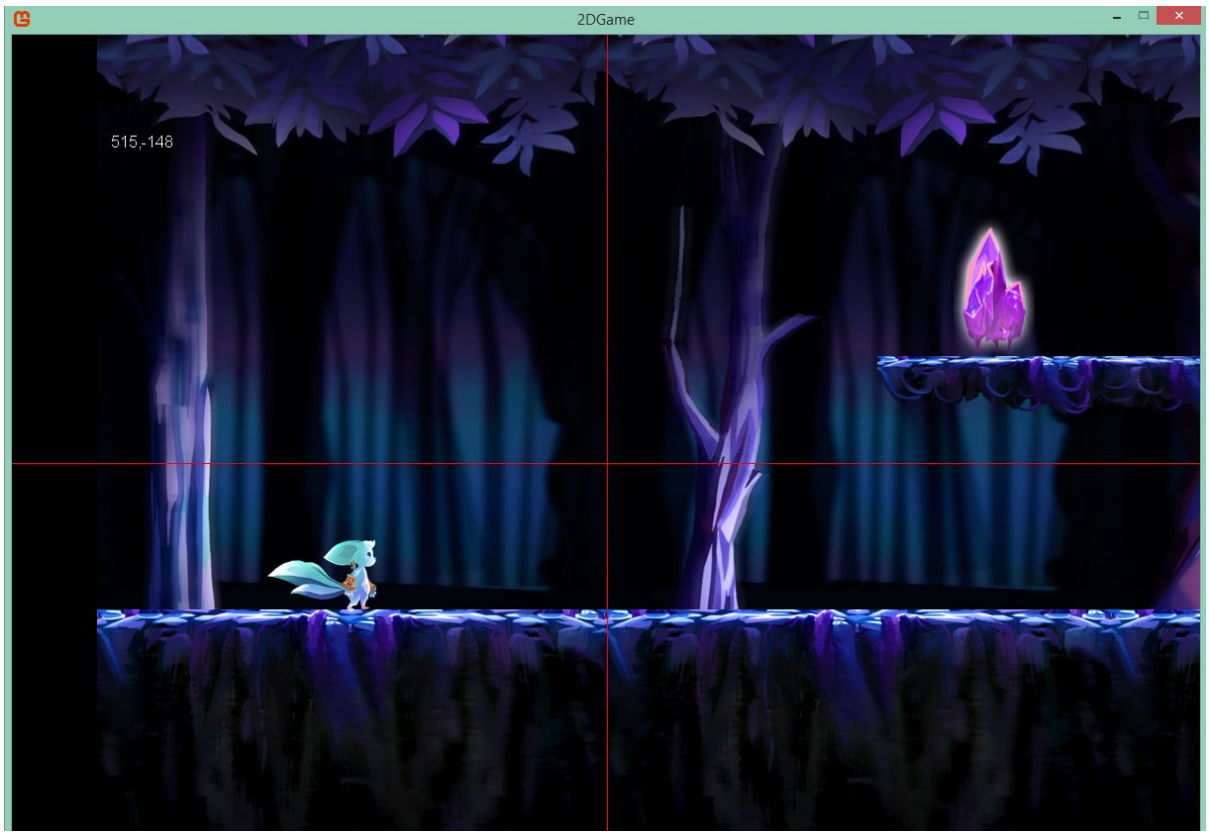
Levels

The Level.cs class is the parent class used to inherit from when creating a new level.

Each level has a Initialise(), Update() and Draw() method. Initialise() sets up the level, Update() is used to update the various objects in the level that require updating and Draw() draws everything to the screen.

Fennex currently only has one level DarkForest.cs. Unfortunately I didn't have time to make a level editor which would have simplified the process of placing objects in the level. However I did implement a level editing mode which

displays the world position of a pixel at the intersection point of the cross hair and also allows me to take control of the camera and move it around using the arrow keys.



This helped with getting the coordinates for placing objects in the world.

For drawing the level objects I created three separate lists Background, Middleground and Foreground which I used to control the order that objects in the world were drawn. This prevents things like background images from being drawn over the player and other level objects.

I also created a list of platforms which I use to check for player collisions, and since MovingPlatform.cs inherits from Platform.cs, this list also contains moving platforms which are checked for a collision.

Crystals and Moving Platforms

For crystals and moving platforms I used the Vector2.Lerp() function to calculate a new position for the object based on how much time had passed and how much time the object should take to complete a trip to the end position.

I also store this change in position and use this value inside the player class. Adding this value to the player's position allows me to move the player with

the platform when the player is standing on a platform.

Extra Information:

Assets

- **Textures**

The texture assets were purchased from GameDev Market online.

The creators profile can be viewed here:

https://www.gamedevmarket.net/member/maz/

I purchased the assets under the “Pro License”. More information on the following here:

https://www.gamedevmarket.net/about/licences/

Assets Purchased:

-“Environment Pack”

-“Magical Creature”

- **Music**

The music track was downloaded from:

http://www.freesfx.co.uk/

Licensing terms can be found here:

http://www.freesfx.co.uk/info/eula/

Assets:

- “shock_wave.mp3”

Limitations/Improvements:

Collisions

Currently the collisions are checked directly against the players animation texture. However this causes a collision to occur when Fennex’s tail is colliding

with a platform which can cause him to hang onto a platform with his tail. It also means that he isn't able to fall down into gaps where spikes are the way he should.

This can be fixed relatively simply by generating a second set of textures which are the characters alpha maps, which contains a black texture that is compared for the collisions where the tail is removed. Again, time was a factor here and I didn't quite manage to fit in this improvement.

Level Editor

I would also like to add a level editor so that I don't need to manually position objects in the level as it is not scaleable to continue to create levels manually and looks poor quality when assets are procedurally placed (note the flowers in the cave and the trees in the forest).

Camera

I also received feedback that people found the camera tracking a little odd and would have preferred that the camera position itself so that Fennex was always in the centre of the screen to give a better idea of the upcoming obstacles and platforms.

Enemies

Unfortunately the artist I purchased the assets from did not have any enemy sprites. It is my plan to contact him in the near future to see if he would be interested in creating some enemy sprites for the game.

User Interface & HUD

Due to time constraints I only implemented a very basic user interface with a start screen. This could be greatly improved with things like a controls screen that explains the controls to use Fennex.

Code Design

Some of the game features like checkpoints, the player state and the drawing of images could be improved.

The player state lookup does not scale very well when Fennex starts to have more states and animations he could be in.

The drawing of images could also be redesigned once a level editor is introduced, this would allow me to automatically set what layer the object should be drawn at rather than trying to manually draw the game objects in the correct order.

Levels, Life & Game Over

Currently Fennex has no life system which allows the player to try to complete the level any number of times they like. Fennex will be reset to the last checkpoint obtained if he dies. Given more time these things should be implemented.

The level is completed when Fennex collects all of the crystals and run past the end icon:

I would like to add a counter to the screen to show how many crystals out of the total amount of crystals Fennex has collected, and add some way to make the end of the level more clear, like a door or portal opening.