Asteroids

# Player ship selection

## Related Classes

* Menu.cs
* Program.cs

## Overview

In the base code, the player ship selection was the selection of the bitmap for the Player. Menu would store the selection as the file path of the bitmap.

Development would alter the list of bitmaps available to be loaded from a JSON file. Containing a simple array of strings that refers to bmp file paths. As well as the modularisation of code to be able to handle a variable number of ships.

## Considerations

* The menu currently can handle a variable number of ships to display and be selected, however having too many will cause them to overlap over each other.
* Player 1 selection is removed from list so Player 2 cannot select it. This can be undone, but as is the effect is that the selected ship will not be included in the drawn list.
* There is no back function, but currently in development the backspace functions as a complete reset to the menu section

## Future Developments

The current code is not set to handle variant stats for a given ship. To handle this, the passed value taken in program.cs needs to be shifted from the file path to a complete player object, or json to load the Player's game values (including the alteration of player class to handle variable stats). As well as altering the ship json list into a more complex json object, to hold the values.

The idea of having a shifting selection where the current selected ship would be centered and smoothly moved around or scaled up was considered. However at the time, my understanding of sprites was minimal and focused on other things.

Key Callback were originally envisioned to be utilised here to refactor the current system. However i delayed proper implementation of a suitable framework in the following project, which still needs to be adapted to a proper standalone class.

# Main menu

## Related Classes

* Menu.cs
* AstGameObj.cs

# Overview

The original menu used only the option selections drawn onto the screen.

Development shifted their position lower to introduce a title text, which shifted in brightness per loop as well as background. Background large rocks were added, utilising my sprite handler class, which contained a json loading and vector functions. As the base sprite class will translate its velocity with its rotation value, the rocks needed to redirect its velocity every time it updated its rotation, thus the AstGameObj.cs has these functions built into it.

## Considerations

* While the rocks loaded in the main menu are the same sprites used for the enemy rock large in the game proper, they are not enemy objects. Their random spawn details differ from them due to using my own randomising function. The differences include that its pathing includes every section of onscreen, whereas the enemy random will select a point inside the centre half of the screen to intersect, as well as the variance of velocity and rotation speed.
* Currently the temporary variable name for AstGameObj.cs is "aso" while it should be "ago" to be consistent.
* The menu class is quite cluttered, not as much as some others, but can benefit from sectioning certain codes into their own file. (Shifting title text, background handler)
* Background rocks currently reset after counting how many frames they have been positioned offscreen

## Future Developments

The menu only consists of the singular font used throughout the entire game, proper images may make it look better, a background could be something to consider as well.

Other objects could be added in to spawn in the background instead of only large rocks.

Collision physics were considered as well, however I assumed at the time that SplashKit had in-built functions to handle this, and was proven wrong.

As in the prior section, Key callback re-factoring may be performed here.

# Frame Tick Logger

## Related Classes

* FrameTickLog.cs
* Program.cs

## Overview

This object is designed to retain the last X number of milliseconds needed to complete a game loop. It has several functions to be utilised in the loop "Reset", "Update" and "Draw".

Reset should be called at the start of the loop, and Update at the end. In order to give it the entire loop time.

Draw will show the average milliseconds of frames and the longest time retained in its list.

Hertz (Hz) can be roughly converted by Where t is time in milliseconds, and f is the frequency.

## Considerations

* As described above, the displayed values are in milliseconds, and need to be converted into Hertz to be easily quantifiable. However the value 17 is roughly equal to 60Hz (16.666r milliseconds)
* Removal of the calculation of longest frame can be done if decided to utilise too much processing, however it did not seem to make a significant difference in its absence on the RPi 3B+, nor did the lack of the logger itself.

## Future Developments

Currently the displayed values do not have their dimensions marked, which being in milliseconds can cause confusion as to its proper value. This should be changed to add either the ms to the string or altered into a proper Hertz frequency value.

Increasing the complexity of this object can go against its idea of being a minimalist performance metric, but, if necessary, can be expanded to display other metrics if known.

# AstGameObj.cs

## Overview

This class is meant to expand on the sprite class from SplashKit. In its current form found in Asteroids, mostly it handles the sprite's rotation, by being able to retain a continuous rotation speed and translating the velocity of the sprite every rotation update, thus keeping it window-relative.

It has a JSON loading function, which can add to consistency of creating sprites. The JSON form it is designed for is as follows

* "name" – name of bitmap
* "bmp\_fp" – file path of bitmap
* "cell\_details" – array of numbers, used to set up bitmap
* "anchor\_pos" – array of numbers, used to change sprite anchor point
* "anim\_fp" – file path of animation script
* "start\_anim" – animation name to immediately run on creation
* "mass" – number used to set up sprite mass value
* "values" – JSON object, used to hold any other additional information

Most of the fields are self-explanatory, however the "values" field should be considered if you wish to add any other specific information to a type. Sprite has the ability to hold key-value pairs like JSON, but the value is limited to numbers, thus the values JSON found in this object is more adaptable especially if string or objects are to be stored.

## Considerations

* Avoiding the use of this object is easy, as it only serves to handle a singular sprite object. However its designed to be a consistent function to load a sprite from a singular source file.
* The values JSON can hold other file paths, names or complete object information (needs conversion functions), if this feature is necessary.
* The "bmp\_fp" and "cell\_details" are unnecessary if the bitmap name exists and has been setup such as in a resource bundle. "anchor\_pos" "anim\_fp" and "start\_anim" need to be setup with the sprite as they are not bitmap relevant.
* Mass has no intrinsic relevance to any existing SplashKit function, however I did refer to it in a collision physics function outside of this game.

## Future Development

"bmp\_fp" and "cell\_details" can be removed if the bitmap name is properly configured beforehand. As at the time I was not aware of resource bundles in SplashKit. "anim\_fp" can similarly changed to just the animation name, if loaded.

The idea of this object is to contain common functions that pertain to the singular sprite instance. Other classes that intend to use this object in a 1-1 relationship should consider inheritance. Such inherited objects should be designed to contain more behaviour specific functions to the type of object it represents, and define the "values" field keys if they exist.

# JSON Level

## Overview

This is a basic test level that utilises json to store a level's details. It currently can only spawn rocks of S/M/L type in a random pattern. A title is also displayed for a number of frames stated in the json.

Currently the json only handles the tick time until next spawn, number of rocks to spawn, type of rock and speed of rock.

## Consideration

* There is not much completeness to this level loader, but it's not dependant on a switch structure. As it is not hard-coded, it can be easier to alter, JSON values can be edited after compilation unlike the other levels.
* It can only do a randomised spawn, no capability to set specific X/Y position, or special patterns
* Currently cannot spawn bosses, and the level complete is hard-coded to switch to level1 after no more enemies remain in the list when the JSON finishes iterating the spawn waves.

## Future Development

There is a lot of functionality that should be added to this class in order to reach parity with the hard-coded levels. Such as set spawn patterns or positions, boss types and other scripting ideas that can be called on by the JSON.

Currently the timer of the level is set to reset every time a wave spawns. Thus it is not a absolute level timer, just a counter between the waves. The JSON values are also setup to be how much time after the previous wave should pass before spawning.

Resource bundle can set up timers as well. Due to this, consideration of making the level timers common to a single timer should be decided on.

# Main()

## Related Classes

* Menu.cs
* Game.cs
* Gamesetup

## Overview

The main class is the start of the program contains primary game loop, initialises the games and menus setup.

* Runs Game setup
* Creates Menu object
* When game starts creates the Game Object
* Contains Splash kit clear screen and refresh procedures.

# Game Setup

## Related Classes

* Main()

## Overview

This class controls initial game setup by reading a JSON object to set a number of properties if the JSON object is missing a new one is created.

The Jason File set the following Parameters

* screen\_width
  + There resolution of the game window width
* screen\_height
  + There resolution of the game window height
* Fullscreen
  + Whether the game should load in full screen or windowed mode
* no\_boarder
  + if loaded in windowed mode, should there be a border
* gameScale
  + Multiplier that adjusts onscreen objects positions and movements
* Game\_bundles
  + Loads the chosen game assets bundle pick the appropriate bundles of the game resolution

There is also a method Keylook up that can be used to set the key configurations by JSON file. (Controle.json) This method looks up the corresponding command from the JSON file and return the matching splash kit keycode.

### Instructions & considerations

Game performance was subpar on Raspberry Pi at the original resolution of 1600 x 900 introduction, or additional scaled games assets were introduced. To play the game at a resolution of 960 \* 540 (a 40% reduction) game scale should be set to 0.6, which multiplies values across the game by 0.6, reducing them by 40%, and the game bundle gamebundle\_0.6.txt should be loaded, which has bitmaps that have been reduced by 40%.

# Game.cs

## Related Classes

* Player.cs
* Level.cs

## Overview

The game class is responsible for tracking most game's functions, creating the player objects, and level objects; most interactions are managed from here as well

### Overview of Methods

* Game Constructor
  + Initialises the Player objects one or two depending on the number of players
  + Sets the game level
  + Creates Sprite packs
* Next level
  + Change the game level to the level provided
* Players
  + Returns the list of active players
* Draw
  + Is the main drawing procedure
* GameOver
  + Original game ending process, has been deprecated and can p[possibly be removed now.
* GameEndCleanup
  + Resets the game stated status which will reload the menu on next loop
  + Releases sprite packs
* Draw Game Over
  + New drawing procedure for game over
* Handel Input
  + Calls the handle input for each Player
* Updates
  + The main game update procedure
  + Checks to see if players are dead if not calls
    - Player updated procedure
    - Hit check against Player
  + If all players are dead call game over
  + Updated all sprite packs
  + Hit check for enemies that can shoot
  + Check if game has ended can call game end cleanup
* HitCheck(Player player)
  + Checks to see if a player has collided with or shot an enemy
* HitCheck(Enemy enemy)
  + Checks to see if a player was hit by an Enemy shot

# Player.cs

## Related Classes

* Score.cs
* Shooting.cs

## Overview

Controls a players movements and drawn positions on the screen.

### Overview of Methods

* Player constructor
  + Sets the game window and player variable, load bitmap image for Player.
  + Calls Respawn
* Respawn
  + Resets the Plaer to the correct staring position on screen and starts and \_InvulnerableTimer
* Killed
  + Sets the Player is dead to ture
* Draw
  + Draws the pler bitmap on screen
  + If invulnerable, makes them flash
* Rotation
  + Calculations the angle of rotation for the by the change supplied.
* Move
  + Moves the player bitmap XY by the speed supplied
* HandleInput
  + Calles the relevant player controls
  + Check if the Player is out of bounds
* Player1Controls
  + Controle schema for Player 1
  + Calls corresponding action
* Player2Controls
  + Controle schema for Player 2
  + Calls corresponding action
* OutOfBounds
  + Checks if the ships XY has moved offscreen and moves them to the corisposing position on the otherside
* Shoot
  + Adds a new shot to the shtos list
* Updates
  + Removes Shots once off screen.
  + Resets invulnerable timer
* HitBMP
  + Returns the players ship bitmap for collision detection
* HitCheck
  + Checks if the images collision circle intercets and enemies HitCircle
  + If enemy has a Collision sprite set check if bitmap intersects
  + Tells the enemy a player hit them (this allows the enemy to take the appropriate action and return a data Tuple with the corresponding action, i.e. take one life)
  + Call shots hitcheck
    - If the enemy hit by a shot, remove the shot
    - Tell the enemy it was hit by a shot
    - Enemy returns Tuple i.e. Score and number

# Score.cs

## Overview

Tracks, controls and displays a player's Score and the number of lives remaining.

### Overview of Methods

* Score Constructor
  + Starts a timer that increments the Score
  + Loads the heats bitmap
  + Sets starting variables
* Start Timer
  + Starts the Score Timer
* Pause Timer
  + Pauses the Score Timer
* Is Timer Paused
  + Check if the score timer is paused
* Resume Timer
  + Resume Timer
* Draw
  + Abstract method for inherited classes
* DownLife
  + Reduces a player life by one
  + Sets the Player to dead once no lives are left.
* ScoreUp
  + Increase score by the provided amount

## Inherited Classes

* Player1Score
  + Constructor
    - No additional parameters usees the base constructor
  + Draw
    - Draws the players score and number of lives with graphic hearts in the correct location for Player 1
* Player2Score
  + Constructor
    - No additional parameters usees the base constructor
  + Draw
    - Draws the players score and number of lives with graphic hearts in the correct location for player 2

# Level.cs

## Overview

Contains code for loading and coordinating events taking place during the level. This includes tracking numbers and type of enemies. Levels are an inherited class. These generally spawned different enemy types based on timing or script

### Level Class

* Constructor
  + Sets some vaibles, level complete bitmap and Timer
* Draw
  + Calls the draw procedure for each enemy
  + Calls level complete draw procedure
* Level Complete
  + Starts level complete timer
  + Sets the next level
* Level complet Draw
  + Draws the level complete timer
  + Starts the next level once level complete times is finished
* Create Enemy
  + Spwans the corresponding rock type based on the supplied variables or uses defaults
* Spawn Tripple (may be depreciated)
  + Spans a triple set of small rocks used when a large rock is destroyed.
* Spawn Rock Wall
  + Spawns a rock wall of the supplied type from the supplied direction
* Update
  + Calls the Enemny update procedure
  + Checks if an enemy is offscreen or dead and sets it to be removed from the enmeny list
  + Check if Tripple Rocks need to be spawned
  + Adds Enemies in the temp spawn (new triple rock generation process)
  + Removes Enemies in the Kill Enemy list.
* Rock Random Spawn
  + Spawns a random rock type
  + Spawnable Rocks types and be controlled by an enumerator
  + Speed of the Rock and spawn rate can be set or use defaults

### Level 1 (Inherited class)

* Constructor
  + Sets up level timer
  + Sets up font
  + Clears any existing enemies
* Draw
  + Draws level name for the first 1.5 seconds
  + Then calls based draw procedure
* Update
  + Contolres the spawning of rockes for the first 55 seconds random rocks spawn with increasing speeds, and the last 20 seconds increased spwan rate
  + 55 second 4 large rockes are spwanred fomr each side
  + At 60 seconds, the boss is spawned

### Level 2 (Inherited class)

* Constructor
  + Sets up level timer
  + Sets up font
  + Clears any existing enemies
* Draw
  + Draws level name for the first 1.5 seconds
  + Then calls based draw procedure
* Update
  + First 15 seconds Random rock spawns at speed 4
  + At 15 seconds spawns 5 of the new blue rock type
  + For the next 15 seconds, spawn all rock types at an increased rate
  + 35 seconds spawn small rock wall
  + 36 seconds spawn blue rock wall
  + 37 seconds spawn large rock wall
  + 39 – 41 seconds spawn 3 small rock walls
  + Spwan boss 2 ( three motherships)
  + Complete level once boss 2 defeated

### End Game (Inherited class)

* Constructor
  + Sets Timer, Font, game
  + Clears enemies list
  + Pauses player score timers
* Draw
  + Displays game complete message
  + Once 10 seconds have elapsed calls GameEndCleanUp

### Debug Level(Inherited class)

* Constructor
  + Sets up level timer
  + Sets up font
  + Clears any existing enemies
* Draw
  + Draws level name for the first 1.5 seconds
  + Then calls based draw procedure
* Update
  + Used to create game elements you want to test.

# Enemy.cs

## Overview

Contains code for spawning each enemy type, including bosses

## Enemy Class

* Constructor
  + Set's variable defaults and sprite pack to Enemies
* Hit Circle
  + It is Virtual, so inherited classes can override
  + Returns default hit circle
* HitSprite
  + It is Virtual, so inherited classes can override.
  + Default returns null
* Draw – Abstract no implementation
* Kill Test
  + Sets Is dying to true
* Rotation
  + Calculates angle of rotation for bitmaps
  + May be deprecated
* Free sprite
  + It is Virtual, so inherited classes can override.
  + Has no implementation but is virtual, so implementation is optional
* Update
  + It is Virtual, so inherited classes can override.
  + Calls rotation
  + Increases is dying count (function may be deprecated)
  + Updates X,Y by velocity X,Y
* HitBy (Player)
  + It is Virtual, so inherited classes can override.
  + If a player was hit by a enemy that's not dying, their life is reduced by 1
* HitBy (Shot)
  + It is Virtual, so inherited classes can override.
  + If is dying doesn nothing
  + Call free sprite of the shot
  + Returns score of 10
* Is Off Screen
  + It is Virtual, so inherited classes can override.
  + Returns if XY off screen (may be deprecated)
* Rnd form PT
  + Determines a random spawn location using random number generators
* SetCourse
  + Sets a random direction within a 200-pixel area in the centre of the screen

## Rock Large inherited class

* Constructor
  + Loads a Ast Game object based on JSON file
  + Sets rotation speed
  + Calls set course
  + Sets XY, velocity and height
* Hit Circle
  + Returns a hit circle based on the sprites' collision circle
* HitBy Player
  + If not already dying, start explode animation
  + Return base hitby
* HitBy Shooting
  + If not already dying, start explode animation
  + Return base hitby
* Is Offscreen
  + Returns Sprite Offscreen procedure
* Update
  + Updates XY to Spriote centre point XY
  + If the animation is set to explode and the animation has ended
    - set spawn small rocks true (this is checked in the level class)
    - Starts the explode 2 animation
  + If the animation is set to explode2 and the animation has ended
    - Set is dead
    - Free Sprite
* Draw
  + Inactive code for debugging

## Rock Med inherited class

* Constructor
  + Loads a Ast Game object based on JSON file
  + Sets rotation speed
  + Calls set course
  + Sets XY, velocity and height
* Hit Circle
  + Returns a hit circle based on the sprites' collision circle
* HitBy Player
  + If not already dying, start explode animation
  + Return base hitby
* HitBy Shooting
  + If not already dying, start explode animation
  + Return base hitby
* Is Offscreen
  + Returns Sprite Offscreen procedure
* Update
  + Updates XY to Spriote centre point XY
  + If the animation is set to explode and the animation has ended
    - Set is dead
    - Free Sprite
* Draw
  + Inactive code for debugging

## Rock Small inherited class

* Constructor
  + Loads a Ast Game object based on JSON file
  + Sets rotation speed
  + Calls set course
  + Sets XY, velocity and height
* Hit Circle
  + Returns a hit circle based on the sprites' collision circle
* HitBy Player
  + If not already dying, start explode animation
  + Return base hitby
* HitBy Shooting
  + If not already dying, start explode animation
  + Return base hitby
* Is Offscreen
  + Returns Sprite Offscreen procedure
* Update
  + Updates XY to Spriote centre point XY
  + If the animation is set to explode and the animation has ended
    - Set is dead
    - Free Sprite
* Draw
  + Inactive code for debugging

## Rock Blue inherited class

* Constructor
  + Sets bitmap, animation script, create sprite
  + Sets anchor point to the centre of sprite (point where sprite rotates)
  + Sets health to 5
  + Sets XY, velocity and height
  + Calls set course
* Hit Circle
  + Returns a hit circle based on the sprites' collision circle if not dead
  + If dead, return a small circle off screen
* Update
  + If dying increase dying count
  + Call Rotation (change rotation angle)
  + Rotate the sprite
  + Move the sprite to new XY
  + If dying animation is set and ended IsDead = true
* HitBy Shooting
  + If is dying return nothing
  + If health is more than 0
    - Reduce rock health by 1
    - Start animation hit
    - Set temp score to 10
  + If health is 0
    - Set is dying to true
    - Start dying animation
    - Adds 50 to temp score
  + Return Score with temp score value

* HitBy Player
  + If dying return nothing
  + Et dying to true
  + Start dying animation
  + Return life -1
* Free sprite
  + Frees the sprite
* HitSprite
  + Returns Sprite
* Draw
  + Empty procedure, encase it’s called

## Boss1 inherited class

### Overview

Creates level one boss with capabilities to shoot downwards with three small circles and fire energy ball at the Player.

* Constructor
  + Setup Boss bitmap
  + Setup Animation script
  + Create animation Shield up
  + Initialtise variables
    - Ship health
    - Shiled Health
    - Game Phase
    - Position
    - Set opt to boss animation
  + Create timers for calculating shot timing
* Draw
  + If Shield is up
    - and flash set to true call Sheild Flash animation
    - And flash false Shield up animation
  + If health less than 10 Critical animation
  + Else shield down animation
  + Call draw procedure for each shot
  + Draw the boss with animation
* Update
  + Phase Start
    - Moves ship down from off screen
  + Phase Mid
    - Moves the boss left to right, moving up and down
    - Calls small shots based on timer
    - Calls Red Energy Ball based on timer
  + Phase end
    - Moves the boss left to right, moving up and down
    - Calls Red Energy Ball based on timer
  + Case Dead
    - When the animation ends and the Shots offscreen boss is dead
  + Moves pahses baed on sheilf and ship health
  + Checks if shots are offscreen and kills them
* Free Sprites
  + Free shot sprites and kill shots
* HitBy Player
  + Return life -1
* Hitby Shot
  + Reduce Ship or Sheild health
  + Return score
* Hit Circle
  + Returns three small circles covering an area under the image
* Hit Check Player
  + Calls player hitcheck for each shot
  + If the Player is not invulnerable life -1
* Shoot Small
  + Created three small shots from the bosses position
* Red Energy ball
  + Creates a red energy ball and shoots it at the Player if they are alive.

## Boss2 inherited class

### Overview

Creates level two boss which is three ships (they are another inherited class). These ships move in a infinity pattern and fire a large red laser downwards. If a player crosses a threshold, i.e. try hide above the ships they will each fire a red energy ball at the player ships very quickly.

* Constructor
  + Set Boss bitmap
  + Set animation
  + Set phase and timers
    - One timer is used for the movement calculation
  + Set the kill threasehold
  + Set the starting position for each ship (\_time.add)
* Draw
  + Call draw for each shot
* Update
  + No Implementation
* HitBy Player
  + Return life -1
* Red Energy Ball
  + Creates red energy ball

## SmallShip inherited class of Boss 2

* Constructor
  + Creates sprite for ship
  + Sets health
  + Starts animation shield up
  + Creates timers for red energe ball and shooting
  + Sets \_time based on ship number
    - This sets the position of the ship in the [Lemniscate of Bernoulli curve](https://en.wikipedia.org/wiki/Lemniscate_of_Bernoulli#:~:text=In%20geometry%2C%20the%20lemniscate%20of,and%20to%20the%20%E2%88%9E%20symbol.)
* Update
  + Calcualtes the XY position for the ship based on [Lemniscate of Bernoulli curve](https://en.wikipedia.org/wiki/Lemniscate_of_Bernoulli#:~:text=In%20geometry%2C%20the%20lemniscate%20of,and%20to%20the%20%E2%88%9E%20symbol.)
  + Sets the ships XY
  + Based on timer shot ship moving and fires laser
  + If ship health is gone play critical damage animation and set isdead true
  + Call PlayerUpdate, ShotUpdate, Laser Update
* PlayerUpdate
  + Checks if the Player if above the kill thresh hold and starts shooting red energy balls
    - These fire at both players in two player mode.
* Fire Laser
  + Sets that the Laser is fireing and the frame count
    - Currently, the Laser is broken up into 40 frames so it can be spawned in small sections.
* Laser Update
  + Calculates the position of the Laser
  + Add laser shots passing in the correct frame
* ShotUpdate
  + Kills shots and lasers if they are offscreen
* Free Sprites
  + Call free sprite on all shots and lasers
* HitBy Shooting
  + Reduced ship heath when hit by a shot flash shield
  + Return score
* Hit Circle
  + Returns three small circles covering an area under the image
* Hit Check Player
  + Check if the shot hit Player
  + Check if Laser hit Player

# Shooting.Cs

### Overview

Contains Code for all the shot types

## Shooting Class

### Overview

Basic setup for a shot

* Constructor
  + Implemented in inherited classes
* Freesprite
  + Does nothing but allows inherited classes to free their own sprite
* Draw
  + Draws a white filled circle on screen with XY & radius
* Update
  + Updated the XY based on velocity
* IsOffcereen
  + Returns if XY is off screen
* Damage
  + Returns default damage for a shot

## PlayerShot inerhited class

### Overview

Class for player's basic shot

* Constructor
  + Set Starting values
    - Anlge Based on player angle
    - The player that fired shot
    - Speed
    - Size (radius)
    - Direction
* HitCheck Enemy
  + Check if temp circle intersects with the enemy hit circle
* HitCheck Player
  + Returns False

## PlayerShot inerhited class

### Overview

Small Shot type form boss 1

* Constructor
  + Set Starting values
    - Set XY to fromPT
    - Set Direction
    - Set size
    - Create circle at XY
* Hitcheck Player
  + Check if the shot circle intersects with the player bitmap
* Update
  + Updates the Cirecle XY position
* HItCheck Enemy
  + Return False
* Draw
  + Draws the circle (shot)

## RedEnergyBall inerhited class

### Overview

Large animation shot fires by bosses

* Constructor
  + Set Starting values
    - Set XY to fromPT
    - Set direction to player location
  + Set Bitmap
  + Set animation script
  + Create sprite
    - Set Anchorpoint
    - Set position
* HitCheck Player
  + Checks if the sprite has hit the player bitmap
* HItCheck Enemy
  + Return False
* Update
  + Changes the Sprites XY position
* IsOffscreen
  + Checks if sprite is offscreen
* FreeSprite
  + Frees the sprite

## RedEnergyBall inerhited class

### Overview

Laser used by Boss 2

* Constructor
  + Set Speed
  + Set Bitmap
  + Set animation script
  + Create sprite
    - Set Anchorpoint
    - Set position
    - Start animation based on frame number
* HitCheck Player
  + Checks if the sprite has hit the player bitmap
* HItCheck Enemy
  + Return False
* Update
  + Changes the Sprites XY position
* IsOffscreen
  + Checks if sprite is offscreen
* FreeSprite
  + Frees the sprite