

Save the Animals

Samir Mohamed and Shannon Li

Introduction

This game was created as a final project for John Sterling's Intro to Game Programming course (CS3113). Our intention behind the game was to exercise the concepts and tools we learned in the course: flocking, collision detection, gravity, and 3D transformations.

Storyline

In this game, aliens are invading and stealing our animals for experimentation. You must save the animals from the alien lab. You need to rush through each room and collect as many animals as possible before the time runs out. Once you pass the last room, you wake a spaceship alien who chases after you while you try to exit the lab with your collected animals.

Gameplay

This is a platform game. Each level has a number of animals and portals. You need to navigate the level with the portals to collect as many animals as possible. To advance to the next level, you must be standing on the "next level" platform, either when time runs out or after you've collected all the animals.

After you've reached the spaceship alien, you must run back through the levels and hit the "previous level" platforms to retreat to the previous level.

You lose the game if you aren't on the "next level" platform before time runs out.

You win the game if you successfully exit the lab.



You

This is your character. You move with arrow keys (UP, LEFT, RIGHT)

There are two different **platforms**:



Solid

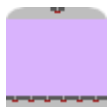
You cannot go through this block



Plank

You can jump through this block if you're coming from below it. You can stand on top of it.

There are various **portals** that alter gravity for whatever passes through them:



Neutral

Sets the force of gravity to the normal amount



Rightside down

Changes the direction of gravity to pull right

**Leftside down**

Changes the direction of gravity to pull left

**Downside down**

Changes the direction of gravity to pull down

**Upside-down**

Changes the direction of gravity to pull up

**Half gravity**

Halves the strength of the force of gravity

**Double gravity**

Doubles the strength of the force of gravity

**Animals**

This is an animal. They may behave differently, but this is what you need to collect.

**Next Level**

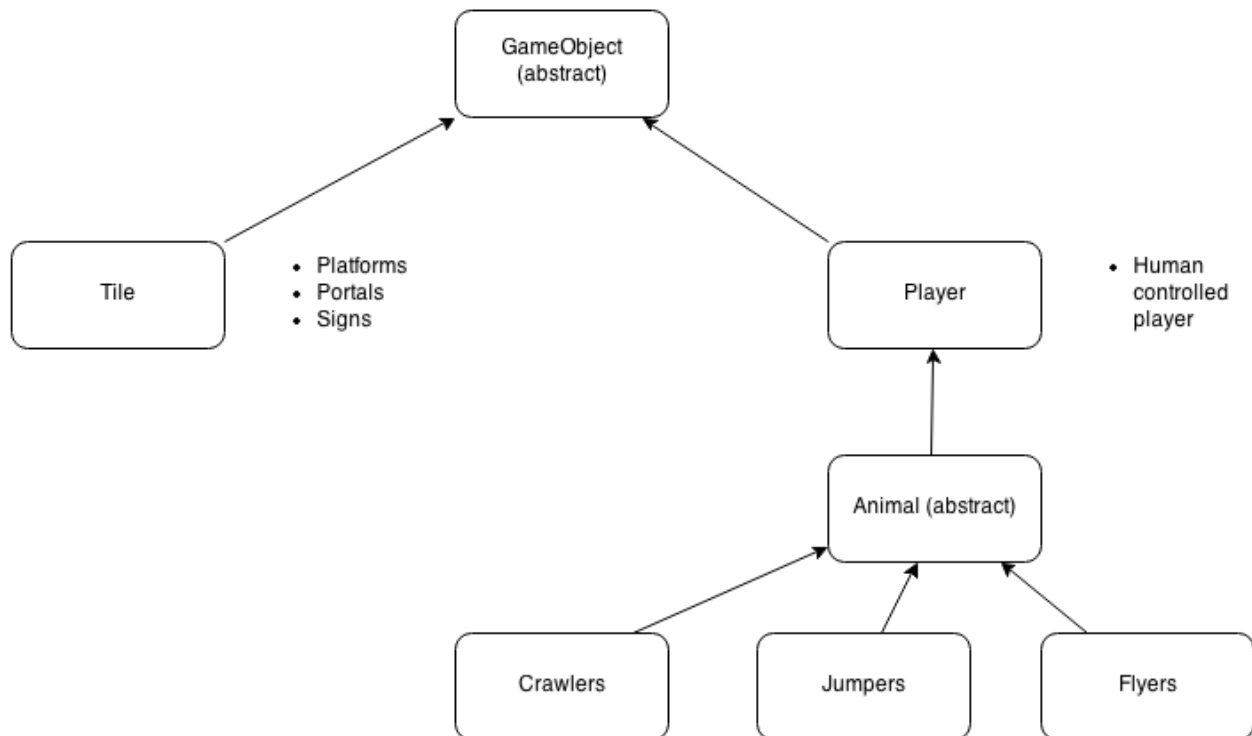
Stand on this sign after collecting all the animals (or right before time runs out) to go to the next level

**Previous Level**

Stand on this sign to go back to the previous level when fleeing from the spaceship alien

Design

Here is an overview of the class hierarchy:



Game Object

GameObject is an abstract class that is the parent of the Tile class and the Player class. Each GameObject has a rectangle “boundingBox”, a portalType1 “pt1”, a portalType2 “pt2”, and a GOType (Game Object type) “type”. Many of the objects’ unique properties will be decided in Game1.cs by looking at the “type”. The X, Y, Height, and Width properties of boundingBox can be accessed directly. The GameObject also has “Intersect” and “Intersects” functions which will check the intersection of the boundingBox to another GameObjects boundingBox.

Portal Types

The portal types, “portalType1” and “portalType2” are enumerated types that will communicate what kind of gravity the GameObject has. The first portal type describes what direction gravity is pulling and the second portal type describes how strong the force is.

Tile

Tiles are a simple child of GameObject. The tiles are created when levels are loaded. The levels are loaded by reading in a text file with 15 lines (rows) and 20 characters per line (columns). The row and column of each character will decide the boundingBox location while the character decides the type. The characters being used are defined in GOType in GameObject.cs. The platforms, portals, and arrow signs are all different types of Tiles.

Player

The player class is a child class of `GameObject` that adds many new properties. These properties allow for gravity and for collision detection during movement. The character that the user plays as is a `Player` object, as are any `GameObjects` that can move.

Animal

`Animal` is an abstract child class of `Player` and inherits the gravity and collision properties while adding animal behaviors. Their behavior algorithm is defined in the “doWander” method.

Player Movement

Player movement is done in the function “handlePlayerMovement” in `Game1`. It handles the keyboard input for the player. It tracks what the changes of the players x and y would be under normal gravity, and then applies it to the appropriate direction depending on the players `portalType1`.

Platform Collisions

The function “handlePlatCollisions” takes a player object and checks for collision with every `Tile` of the current level. If the player is mostly inside a portal, one of its portal types will change. If the player did not land on any platforms, then he will be considered jumping so that gravity will start being applied. This function uses the player’s `portalType1` to make collisions appropriate to the direction of gravity.

Crawlers

Inherit from `Animal`. These animals have a very simple wandering algorithm. They wander back and forth along the platform they are standing on.

Jumpers

Inherit from `Animal`. These animals can jump to avoid the player. They can jump off of the platform they’ve started on and fall to the ones below it.

Flyers

Inherit from `Animal`. These animals are not affected by gravity (only their sprite image is rotated). They implement a flocking algorithm and tend to cluster together, while avoiding the walls of the game screen, solid blocks, and the player.

Challenging Aspects

Collision detection and flocking were the most challenging features to implement.

For collision detection, there were many cases to cover when relocating the player after colliding with a block, especially with the different gravity directions. If a bug was found or a platform property was changed, the change had to be applied differently to all four directions of gravity.

Flocking, although the concept seemed simple to implement (average the directions/locations of the animals within a distance from myself), took a long time to successfully implement. It was easy to leave out some of the important factors, such as using the distance between the boid and myself as a weight for how much to move in their direction.