Getting Started with Alien Banjo Attack

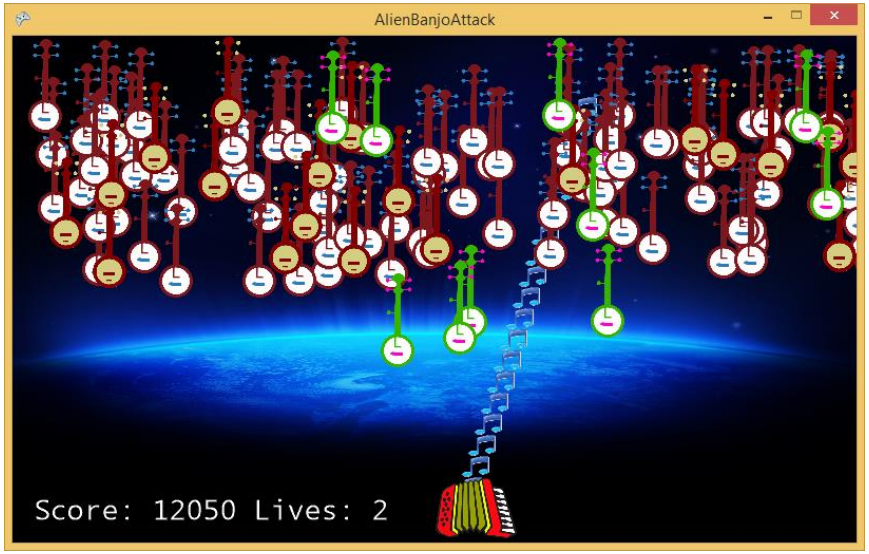
Rob Miles

Department of Computer Science

Alien Banjo Attack Game

- By now you should be well on with the Alien Banjo Attack game
- But just in case you aren't....

Note that if you have solved these problems in a different way there is no reason to ditch your solution in preferences to mine



How to get started

- You don't create the entire game all in one
- The best approach is to build up the behaviours, one simple step at a time
- This is how we can create any kind of complex software system
- So, to start you must decide the sequence of items you are going to build

How to get started

- You don't create the entire game all in one
- The best approach is to build up the behaviours, one simple step at a time
- This is how we can create any kind of complex software system
- So, to start you must decide the sequence of items you are going to build

Any ideas?

My Suggested Sequence

- This is the sequence that I suggest:
  1. Get the player moving around the screen under the control of the player.
  2. Get the first banjo sprite drawn and make it move across the screen and back.
  3. Add the extra sprite types and the note missile.
  4. Create the game behaviour (attract screen, gameplay and game over)
  5. Add the game storage and recovery
- If each of these steps seems too large you can break them down into smaller ones

1. Get the player moving

1. Load and draw the sprite and background textures
2. Position the texture on the screen
   - You can use Rectangle or Vector
3. Get the texture to move
   - Update the position value each time using speed value
   - Detect the edge of the screen and make it move down
4. Get the player to control the texture
   - Update the speed values using the controls
5. Add the collision behaviour for the edges.

2: Add the Banjo

1. Load and draw the sprite textures for the enemies
2. Position them on the screen
   - The enemies can be in random position along the top of the screen
3. Use a class hierarchy to create different types of enemy
4. Use a list to hold all the sprites on the screen

3: Add the extra banjos and the note missile

1. The more advanced banjos are simply child classes of the simple banjo, but with different `Update` methods
2. They are also loaded with a different texture when they are created.

4: Create the game behaviour

1. Make some states to represent the gameplay
2. Add switch constructions to the `Draw` and `Update` methods so that they behave correctly for each game state
3. Add code that changes the state of the game appropriately
   - Start the game when the user presses Start
   - End the game when the player collides with a banjo or a banjo reaches the bottom of the screen

5: Game Storage and Recovery

- The program must store the state of all the game objects when the user wants to pause the game
- You can treat a Sprite in the same way as you would an Account object
- The game will contain a list of sprites which are drawn and updated
- When the game is saved the sprites details are all written into a file
- When the game is resumed this data is reloaded

Simple Sprite Class Behaviours

```
class Sprite
{
    void Draw(SpriteBatch spriteBatch) { }
    void Reset(BanjoGame game) { }
    void Update(BanjoGame game) { }
}
```

- These are the three essential behaviours that a `Sprite` really needs to implement
- The `Sprite` class will also need a constructor that sets it up with its texture and initial position
- Note that the `Update` and `Reset` methods are given a reference to the game the `Sprite` is part of
  - This is so they can tell the game if they are killed etc

Updating the game elements

```
// Get each sprite to update itself
foreach (Sprite s in sprites)
    s.Update(this);
```

- My version of the game contains a list of sprites
- The `Update` method works through the sprite list and updates each one in turn
- Note that the word `this` in the context of the call of `Update` means "the current instance"
  - In this case it is the game that the sprite is part of
  - The sprite can then update the status of the game if anything happens

Saving and loading the sprite positions

```
public void Save(TextWriter textOut)
{
    textOut.Write(SpriteRectangle.X);
    textOut.Write(SpriteRectangle.Y);
    textOut.Write(SpriteRectangle.Width);
    textOut.Write(SpriteRectangle.Height);
    // rest of values written out here
}
```

- The game must save the positions of the items on the screen
- You can do this by treating each sprite as a bank account and adding Save and Load methods that work the same way

Game State

```
enum State
{
    AttractMode,
    PlayingGame,
    GameOver
}
```

- You can use an enumerated type to represent the state of the game as it is being played
- In the example above the state has three possible values
  - Attract mode: show a screen that invites the user to press a button to start
  - Playing game: playing the game
  - GameOver: showing the high score

Game State

```
State gameState = State.AttractMode;
```

- The game contains a variable that holds the state of the game at any given time
  - Above I have called the variable `gameState`
  - It is initially set to the attract mode
- When events happen to change the state of the game this is achieved by updating this variable
  - If the player collides with a banjo, or the banjos reach the bottom of the screen the state would be changed to `State.GameOver`

Using Game State

```
switch (gameState)
{
    case State.AttractMode:
        break;
    case State.PlayingGame:
        break;
    case State.GameOver:
        break;
}
```

- You can use a switch to select the appropriate action in the Draw and Update methods in the game
- If you add extra states (for example high score table) you just need to add the extra state and the cases in the switches that are controlled by it

Attract Mode

- When the game is in “attract mode” something must be moving on the screen
  - randomly drive the player in different directions
  - replay a set of movement instructions
  - You could even replay the previous game
- Attract mode ends when the user presses a key to start the game
  - In attract mode your game must test for this

Play mode

- When play starts everything must be reset for the start of the game
- During play mode the user will steer the player around the screen and shoot at the banjos, who will be moving around and attacking
- If the player hits an alien banjo or a banjo reaches the bottom of the screen the mode of the game is changed to game over

Game Over Mode

- At the end of a game a Game Over screen should be displayed for a few seconds
- You can time this by counting the number of times that Update has been called
- It is called 60 times a second when the game is running
- If you count up to 300 this will give you a 5 second delay

Problem Solving

- Keep each individual step a small one so that it is easy to debug
- Don't "run away" from problems you can't solve to other problems that you can't solve either
- Get help when you are stuck, but make sure that you ask "How do I do this?" rather that "What do I do?"