MegaBoy

A platformer by Stuart Bryson

Game Design Assignment 3

Name: Stuart Bryson
Student #: 98082365

Table of Contents

Game Design

Story

Premise

The protagonist, MegaBoy, must use the power of emeralds to free the innocent creatures from the world dominated by evil monsters.

Backstory & Plot

MegaBoys dog, MegaDog, has been taken captive by MegaBoys evil nemesis, Dr Dangerous. This story is introduced to the player through the use of pre-level graphics. If time permitted, further work could go into in-game cut scenes which would help keep the player immersed. It would allow for further story and plot development throughout the entire game.

Synopsis

We are introduced to the protagonist and antagonist through the story introduction before level one. The player is then immediately dumped into the action of killing enemies and freeing friendlies. Later levels and cut scenes inform the player of exactly how the enemies and friendlies came to be in this situation and why the emeralds are required to free the friendlies. We finish the game with a battle against Dr. Dangerous and once defeated MegaBoy can return home with his dog.

While the story is clichéd, it provides us with all the important aspects of a good story including the protagonist and antagonist wrapped up in a Hollywood 3-act story.

Theme / Setting

Levels start off in a pleasant environment where the evil monsters are starting to take over. The innocent creatures are plentiful while the monsters are few. As the protagonist progresses through the game, the levels become much more dominated by monsters and fewer and fewer innocent creatures can be found. The levels progress from a lush, green and friendly environment to a cold industrial, dangerous one, indicating that MegaBoy is getting closer to his nemesis Dr. Dangerous. Finally MegaBoy must face Dr Dangerous in a one on one battle to free his dog, MegaDog.

Characters

The main character MegaBoy has various abilities including the ability to easily jump around the level, shoot and most importantly pick up emeralds which can be used to free friendlies.

The shadow character, Dr. Dangerous, is not encountered until the end of the game. However, we are introduced to Dr. Dangerous even before the first level and throughout the other levels through the use of in-game cut scenes. This ultimately foreshadows the final battle that must take place in order for MegaBoy to free his dog.

Puzzles / Objectives

Being a platformer, MegaBoy has all the usual puzzles that one would expect. These include navigating a large level of platforms, avoiding spikes, pits and enemies. Enemies must be destroyed with MegaBoys gun.
The more interesting puzzle that MegaBoy presents the player is that of freeing enemies. In order to free an enemy, MegaBoy must collect an emerald and then collide with a friendly. This will set the friendly free from the evil world of Dr. Dangerous. Once all of the friendlies are freed, the player can reach the next level by way of a large emerald at the end of the level.

Design Decisions

There were many design decisions that went into MegaBoy. In fact, a lot of the decisions and ideas never saw completion due to time constraints. However, following are some of the decisions that helped shape MegaBoy into the game it is now.

Sprites

Throughout the game, it was important to have very fluid moving sprites. While the sprites were borrowed from other games including MegaMan and Sonic, quite some time was invested into displaying these sprites in a fluid fashion.

Particular attention was paid to the player character. There are different sprite sets for left and right (as I was authoring on the unregistered version of GameMaker), for jumping and running with and without the gun drawn (no pun intended).

The jump sprites were quite difficult to master. This was due to the fact that the hero will jump different heights depending on the user input. That is, if the jump key is held longer, the hero will jump higher. With this in mind, I could not simply loop the jump animation. Nor could I scale the speed of the animation based on the jump height because the jump height is not foreknown. The end result is pausing the animation at the peak and trough of the jumps. This provides a very believable and fluid motion for the jumps and can be seen by taking small and large jumps.

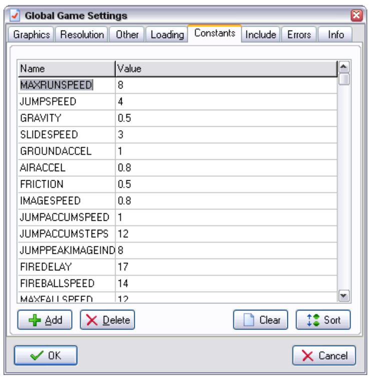
Human Interaction

The most important aspect of a platform game is being able to accurately control the hero or player character.

There are different approaches to controlling the character and most involve directional input and other function buttons. It is likely that MegaBoy will appeal to casual gamers who want a game that is easy to pick up and play for a brief time. MegaBoy uses the arrow keys to control the direction of the character and space to fire. These are extremely obvious controls for a platformer and make it easy for the casual gamer to play.

Further, an easy to control hero needs to not only have easy input controls, but must have well behaved physics. When designing MegaBoy, various third party add ons and scripts for GameMaker were evaluated, however none were determined to be suitable. For this reason, I scripted the player movement using simple rules to control the player. This provides the player with very predictable behavior.

During development of these controls, it became very apparent that various aspects of the movement would need to be tweaked. For this reason, the majority of player movement attributes can be modified using GameMaker's global constants. The constants available for player movement include MAXRUNSPEED, JUMPSPEED, GRAVITY, SLIDESPEED, GROUNDACCEL, AIRACCEL, JUMPACCUMSTEPS and more. Using these constants, I was able to take both informal and formal feedback on these controls to help refine the controllability of the player.



Name	Value
MAXRUNSPEED	8
JUMPSPEED	4
GRAVITY	0.5
SLIDESPEED	3
GROUNDACCEL	1
AIRACCEL	0.8
FRICTION	0.5
IMAGESPEED	0.8
JUMPACCUMSPEED	1
JUMPACCUMSTEPS	12
JUMPPEAKIMAGEIND	8
FIREDELAY	17
FIREBALLSPEED	14
MAXFALLSPEED	12

Other Design Considerations

Another important aspect of the design was to decide how often the player could fire. If the player could simply fire continuously then they would potentially run around the entire level shooting everything. By restricting how often they player can fire, we increase the importance of timing within in the game, and we also encourage the player to play the game sensibly.

Further to this, I needed to decide if and when the fireballs should disappear when they go off screen. If the fireballs were to last forever, the player could accidentally or deliberately kill friendlies and or enemies on the other side of the level. This was undesirable. But if I was to remove the fireballs off screen, should it happen immediately? Take for example the player chasing an enemy off screen. If the player shoots at the enemy just as they go off screen, the fireball then disappears before it kills the enemy, and then when the player catches up to the enemy, the player will be left confused as to why the fireball did not kill the enemy. For these reasons, I decided to give the fireball a life of 100 pixels off screen before it is removed.

When a player comes in contact with an enemy, we want the player to lose health, however we don't want the player to lose health for every frame they are in contact with the enemy. For this reason, I introduced a short time of invulnerability. When the player makes contact with an enemy, they will lose some health and become invulnerable for 30 steps. During this time, the player will flash, providing a visual indication of their invulnerability.

Some informal feedback mentioned that the friendlies and enemies were too similar and it was unclear which creatures the player was to shoot and which ones to collide with. One technique introduced to facilitate this need was to make the friendlies stop and look at the player when they came into proximity with the player. This indicates to the player that this creature wants the players attention or help and therefore must be a friendly. Conversely, we see the enemies change into an aggressive attack move when they are within a certain proximity to the player.

Level Design

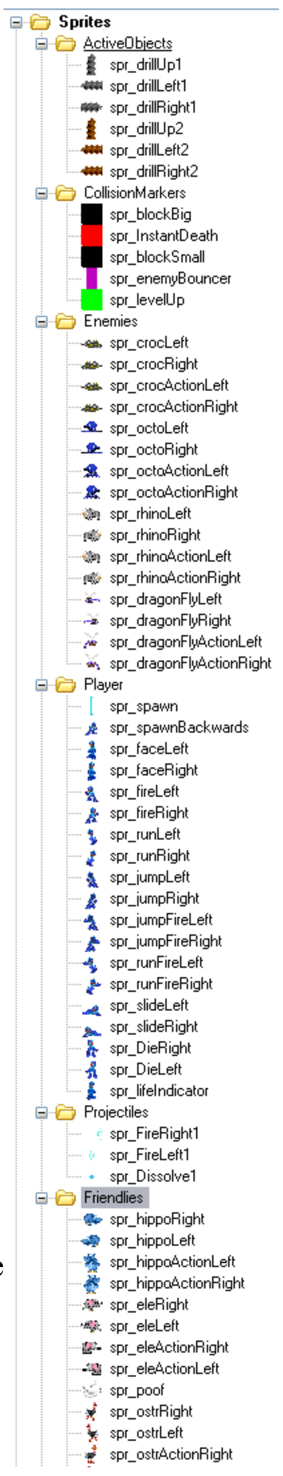
Good level design for any type of game including platformers is a difficult and time consuming task. For this reason, MegaBoy only has 3 levels currently but they have been designed with many things in mind.

The difficulty of each varies in a few different ways. The first level is quite simple. It is designed to introduce the player to the world, get comfortable with the controls and the look and feel of the game. The player is spawned with no impending threats such as an enemy about to attack them. The level contains all the basic threats including air and ground enemies, although these are slow moving, and spikes and pits which will kill the player.

The second level focuses on some more interesting platform challenges including spinning spikes and difficult platforms on the side of walls. We also see an increase in the number of spikes used, particularly the side facing spikes. The side facing spikes were designed such that the player can not jump into any side of the spikes, or even land on the spikes. The player can, however, walk on top of the side facing spikes if they are adjoined to a platform at the same height.

The third level sees less use of spikes and pits but rather increases the challenges to the player by way of more aggressive enemies such as the dragon fly and the rhino. Further levels would continue to combine all of these obstacles together.

Coding and Implementation

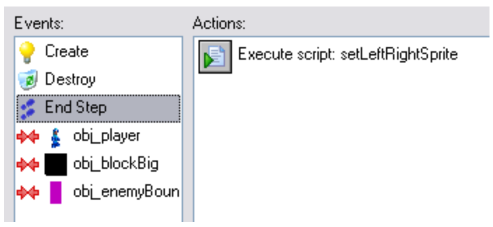


```
Sprites
  ActiveObjects
    spr_drillUp1
    spr_drillLeft1
    spr_drillRight1
    spr_drillUp2
    spr_drillLeft2
    spr_drillRight2
  CollisionMarkers
    spr_blockBig
    spr_InstantDeath
    spr_blockSmall
    spr_enemyBouncer
    spr_levelUp
  Enemies
    spr_crocLeft
    spr_crocRight
    spr_crocActionLeft
    spr_crocActionRight
    spr_octoLeft
    spr_octoRight
    spr_octoActionLeft
    spr_octoActionRight
    spr_rhinoLeft
    spr_rhinoRight
    spr_rhinoActionLeft
    spr_rhinoActionRight
    spr_dragonFlyLeft
    spr_dragonFlyRight
    spr_dragonFlyActionLeft
    spr_dragonFlyActionRight
  Player
    spr_spawn
    spr_spawnBackwards
    spr_faceLeft
    spr_faceRight
    spr_fireLeft
    spr_fireRight
    spr_runLeft
    spr_runRight
    spr_jumpLeft
    spr_jumpRight
    spr_jumpFireLeft
    spr_jumpFireRight
    spr_runFireLeft
    spr_runFireRight
    spr_slideLeft
    spr_slideRight
    spr_DieRight
    spr_DieLeft
    spr_lifeIndicator
  Projectiles
    spr_FireRight1
    spr_FireLeft1
    spr_Dissolve1
  Friendlies
    spr_hippoRight
    spr_hippoLeft
    spr_hippoActionLeft
    spr_hippoActionRight
    spr_eleRight
    spr_eleLeft
    spr_eleActionRight
    spr_eleActionLeft
    spr_poof
    spr_ostrRight
    spr_ostrLeft
    spr_ostrActionRight
```

When developing any Game, I place a strong focus on good code and object design in order that the game can be easily maintained and extended. When the game is easy to extend, I need not waste many hours reworking the engine just to include a new design feature. In other words, good code design enables good game design.

Writing MegaBoy was no exception. A great deal of effort was devoted to developing an easy to extend setup within GameMaker itself. This is through the use of well named sprites and objects, and the high use of reusable scripts and abstract objects.

All GameMaker objects are named with a corresponding prefix such as spr_, obj_ or ctr_ followed by a camel case description of the object. Scripts were freed of this requirement to improve the look of function calls within a script. With a consistent naming convention, authoring and designing the game is much easier.



Events:
- Create
- Destroy
- End Step
- obj_player
- obj_blockBig
- obj_enemyBoun

Actions:
- Execute script: setLeftRightSprite

Another important coding feature is the use of scripts. Many previous examples of GameMaker games have a lot of duplicated actions among different objects. For example, when an enemy or a friendly behaves in a certain way, the sprite must be updated to reflect its behavior. Instead of having actions for every friendly and every enemy that determines which sprite should be displayed, the setLeftRightSprite script works on both enemies and friendlies and uses arguments to alter the script. This is far more efficient, less error prone, and easier to change that duplicating the behavior in each objects event routines.

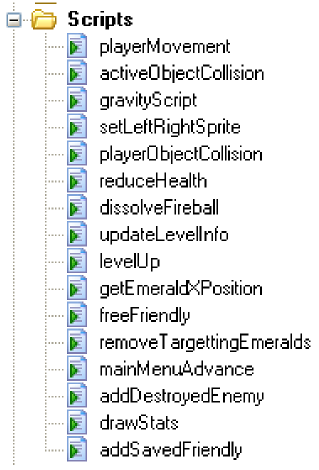
Further, the desire to reduce the amount of code and event routines leads us to the use of abstract classes. In MegaBoy, there are many different abstract classes including emeralds, enemies and friendlies. This enables us to focus all of our object behavior into the parent classes and our base classes become very simple. In most cases, the base classes simply define their sprite and perhaps a speed.

Take the enemies as an example. There is the abstract_AirEnemy which is the abstract class. This defines 90% of the behavior of an enemy including the initial movement, object collisions and setting the sprites. Next there is the abstract_GroundEnemy which inherits from abstract_AirEnemy. It simply adds the gravity behavior. Finally we have the enemies themselves such as the obj_croc which simply sets its initial speed.

Using well defined inheritance such as this enables me to then write simple and powerful scripts. The abstract_groundEnemy, abstract_groundFriendly and the player all execute the gravity script in their step event. This gravity script simply checks if there is a collision below these objects and enables their gravity accordingly.

As mentioned earlier, the game makes heavy use of GameMaker's global constants. This not only helps in game design and tweaking certain behavior, but it also reduces code complexity and removes potential for bugs.

Also mentioned is the playerMovement script written by me. This script contains all the logic needed to control the hero character. The script basically determines that state of the character and the keyboard input and determines what actions to take for that input within the current state. It controls gravity for the player, ground and air acceleration, speed clamping, firing, jump accumulation and lastly setting the player object to display the correct sprite.



Scripts
- playerMovement
- activeObjectCollision
- gravityScript
- setLeftRightSprite
- playerObjectCollision
- reduceHealth
- dissolveFireball
- updateLevelInfo
- levelUp
- getEmeraldXPosition
- freeFriendly
- removeTargettingEmeralds
- mainMenuAdvance
- addDestroyedEnemy
- drawStats
- addSavedFriendly

Some scripts that I evaluated used a finite state machine and set an internal variable to keep track of the players state. This was almost my approach but I decided that the states that I was concerned with were already available to me through other means. The most common 'state' that I used was that of gravity. The players movement differs the most depending on the current state of gravity on the player.

One of the more interesting challenges of the player movement was that of jump accumulation. Jump accumulation simply means that the player will jump higher when the jump key is held down for longer. This provides a simple mechanism where the player can tell the game how high they wish to jump. The code for this basically set an accumulator to 0 when a jump is initiated. This accumulator then increases each step that the players y position is decreasing (ie, moving up the screen). The vertical speed is then increased each step until either the player releases the jump button, or the accumulator reaches it maximum value. This provides a really simple method to produce good jump accumulation.

Fun

Platform games are fun. Ever since I was a child, I always liked to play platform games. It was due to this passion and love of platformers that I chose to develop MegaBoy.

Providing different types of objectives, such as killing enemies and freeing friendlies gives the game some good variation. One could play the game without even killing an enemy, or they could play it with points in mind by killing as many enemies as possible and collecting as many emeralds as possible even if they don't need them all. This variation ensures that different types of players will have fun.

Another aspect of the game that helps make it fun is by authoring parts of the levels that the player does not need to get to to finish the level. This provides those of the diamond and spades player suits, that like to explore and manipulate the game world, the ability to do so.

And lastly there is simply the cute factor of the game. Providing cute little friendlies such as the hippo and elephant goes a long way in providing players with enjoyable and cute things to watch on screen as they battle their way through each level.

Evaluation

When designing a computer game, it is always important to get feedback from your potential customers so that you can refine the game to be the best it can be. The following is a summary of the feedback processes I used when designing MegaBoy.

Story feedback

Before any production even begun, I storyboarded MegaBoy and its main concepts. This was then displayed to 3 or 4 people just to get some initial feedback on whether the game was using a good concept.

The feedback was generally positive. Most thought that the game was very cleched and was not very original however it was a typical platformer that was bound to be addictive and fun. At this stage, I had no differentiating features from other platformers and this is where the concept of 'freeing friendlies', with emeralds the player must pick up, came from. By adding in this feature, I was able to differentiate my game from the other typical platform games found in the market today.

Informal feedback

After some initial prototyping and testing of my game, I got the same 4 people to test my game. At this stage I only had one level playable and there were still quite a few bugs.

The feedback from these informal sessions was quite positive. “The game is very addictive”, “I like the way you are not supposed to shoot the friendlies so you have to be careful”. However there were quite a few suggestions to get the game moving towards a solid platformer.

The most common suggestion was regarding the emeralds. At this stage of development, when an emerald was picked up by the player, the item immediately disappeared from the world and was added to the players emerald inventory. When a player then collided with a friendly, the emerald would animate from the players inventory and fly towards the friendly. Once it reached the friendly, they would both disappear in a puff of smoke.

The comments claimed that the emeralds looked like they were shooting the friendlies from off screen.

I surmised that there was 2 main problems. The first was that the people playing the game were not paying much attention to their emerald inventory, and secondly that the emeralds were animating towards their target too quickly.

To solve the first problem, I modified the behavior of the player collecting an emerald. Instead of it disappearing from the world when the player picked it up, it would animate into the players inventory. This helped people playing the game to be more aware of the inventory.

The second problem was partly solved above as the players were now more aware of their emerald inventory and they now at least knew what was shooting the friendlies. I then determined that by slowing the speed that the emerald animated towards the friendly, it no longer appeared as though the enemy was being shot but rather released by the emerald.

The other feedback was regarding the players movement. Feedback indicated the players jump speed and air accumulation was too slow. After increasing both of these, most thought that the player was much easier to control.

Formal Feedback

Lastly, some formal feedback was received by way of survey. These surveys were then used to further refine the game and also provide ideas for future development of MegaBoy. The results from this survey are attached.

Survey

1. Do you like the concept of killing enemies and freeing friendlies? Does this differ from most of the other platform games you have played.

2. Do you like the player controls? Are they simple and easy to understand? Is the player easy to control?

3. Does the player jump too high, too low? Run too fast or too slow? Change direction easily on the ground and in the air?

4. After playing the game for a short time, do you find it easy to tell the difference between an enemy and a friendly character?

5. Do the levels provide an interesting world to explore or do they get boring? Which level do you like the most?

6. Would the game benefit from more varied player interactions such as switches and moving platforms?

7. Does the game frustrate you in any way? Too many enemies? Platforms too hard to reach? Too many spikes or pits to fall into? If you were to change this, how would you keep the game challenging, yet less frustrating.

8. Would it be good to have enemies shoot at you?

9. Would it be good to have friendlies that help kill the enemies for you?

10. Is the game fun? If so, what makes it fun for you?