CS 402: Mobile Development

Android Game Development

Additional Reading

Drawing Custom Views

Code Labs Canvas Objects

The ABCs of Android game development

Games

Games are hard. We need good:

- Story (beginning, middle, end)
- Gameplay
- Game flow
- Replayability

Use this as a basic checklist for your game.

Prototypes

Some of the best game ideas start as fun prototypes with no story.

The story grows organically based on the fun mechanics.

Hard to make a bad game fun, easy to make a fun game funner.

Story

Every game has some kind of story or theme.

Start with the story. It’ll define the mood, genre, and interaction.

Stories don’t have to be involved, but needs to immerse the player in that world.

Story

It certainly needs an end. Why is the player playing?

Certainly not because the gameplay is fun. Define an objective.

End to a level, and end to a game.

Game Play

Introduce the player to the game, one mechanic.

Introduce them to the controls.

Get them to play more. Always have a call to action to play the next thing.

Game Flow

The game flow is how the game proceeds through it’s story and the player through the levels or scenes.

Good game flow doesn’t jar the player out of the environment or frustrate them without purpose. It proceeds seamlessly from one step to another.

Replayability

You want the player to be able to play the game over and over, especially if all levels share the same type of game play. (ex: Mario)

Components of a Game

Game loop

Game states (playing, paused, scene, loading, etc)

Game logic

Physics

Collisions

User Input

Artificial Intelligence

Game Loop

Infinite but breakable loop that continuously evaluates, reacts and draws the game world.

Works with game state machine to determine how to function.

Game State Machine

Has various states to determine which part of the game you’re in.

Implemented using subclasses of a State class with overridden methods to control basic functionality.

Example

```
class BaseStateMachine{
  fun gameLoop(){
    // empty
  }
  fun onPause(){
    // empty
  }
}
```

Example

```
class MenuState: BaseStateMachine(){
  fun gameLoop(){
    showMenu();
    registerListeners();
  }
  fun onPause(){
    hideMenu();
    unregisterListeners();
  }
}
```

Example

```
class PlayState: BaseStateMachine(){
  fun gameLoop(){
    drawScene();
    registerListeners();
  }
  fun onPause(){
    showPauseMenu();
    unregisterListeners();
  }
}
```

Example

```
fun mainGameLoop(){
  while( this.isPlaying ){
    currentState.gameLoop()

    switch( this.curentUserInput ){
      case KEY_ESC: this.currentState = new MenuState(); break;
      case KEY_ENTER: this.currentState = new PlayState(); break;
  }
}
```

Game Logic

Can become spaghetti code very quickly without proper software engineering patterns:

Singletons

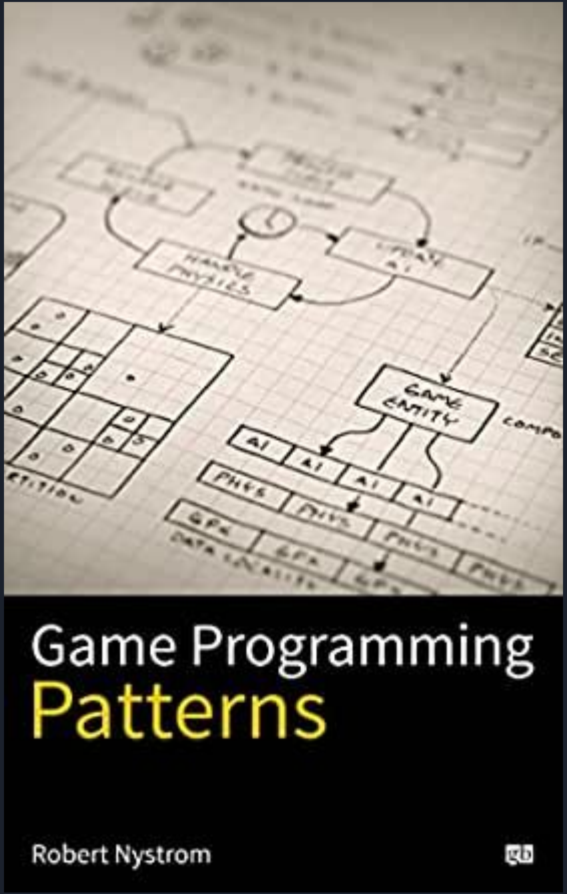
Components

Events

Factories, etc

Game Programming Patterns

Game Programming Patterns - read online



Engineering a Game

Singletons:

Health, ammo, loot, experience, etc can be stored into a player singleton. Can also have one for Game values such as difficulty and music settings.

```
Player.ammo += 50;

Player.health -= 10;
```

Engineering a Game

Singletons:

Breaks down for multiple players.

```
World.getPlayer(player)

.InventorySystem.pickUpAmmo(50);

World.getPlayer(player)

.HealthSystem.pickUpHealth(10);

.HealthSystem.dealDamage(10); // OR
```

Engineering a Game

Factory:

Levels or enemies can be loaded via factories

```
LevelFactory.loadLevel(“Level of doom”,
  Game.playerDifficulty);
```

```
EnemyFactory.spawnEnemy(ENEMY_TYPE_ORC, {30, 100, 60},
{health: 300, level: 32}); // TYPE, LOCATION, ATTRIBUTES
```

Engineering a Game

Events:

Broadcast events to all entities in the game.

```
EventSystem.post(EndGame()) // Players pause, animations
stop, points and pickup cease, etc
```

```
EventSystem.post(EventEnemyDestroyed(player, enemy))
```

Engineering a Game

Use your language and patterns!!

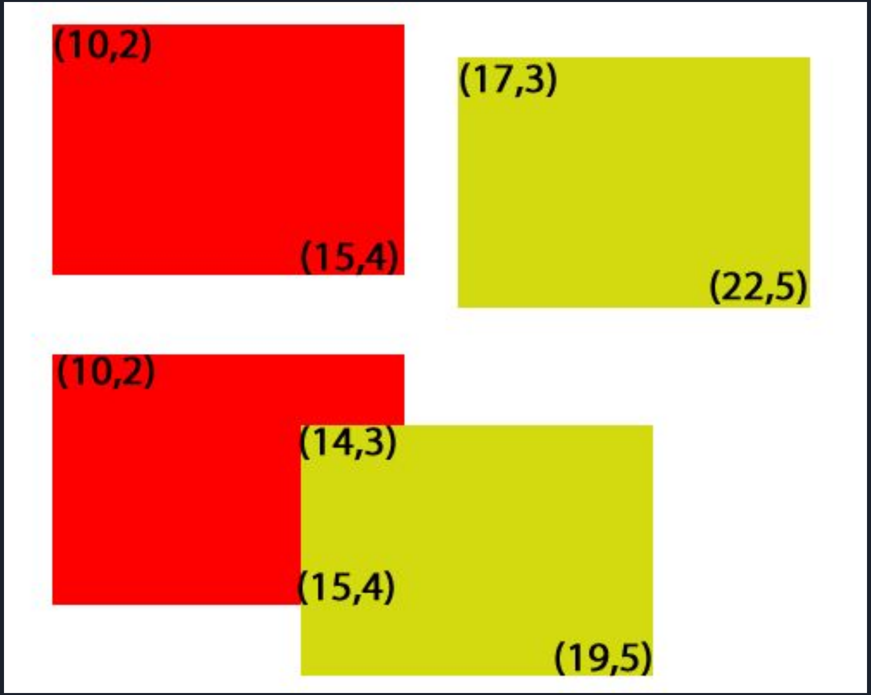
There is almost nothing common, that’s more complex than a game’s logic.

Physics

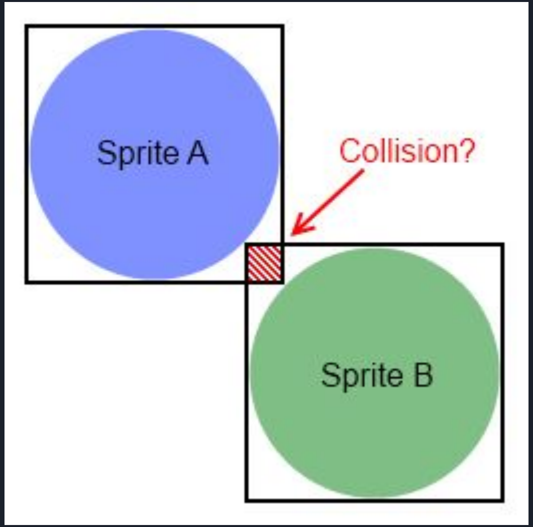
Almost all games have physics to some degree.

Use a 3rd party physics engine for ease of development. It’s also already tested.

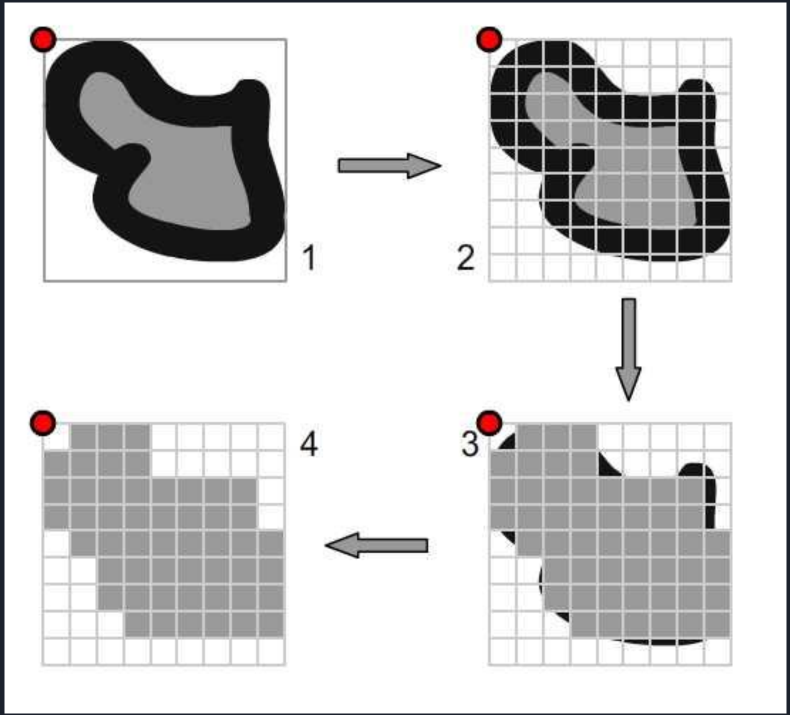
Collisions



Collisions



Collisions



User Input

On mobile, we have:

Tap

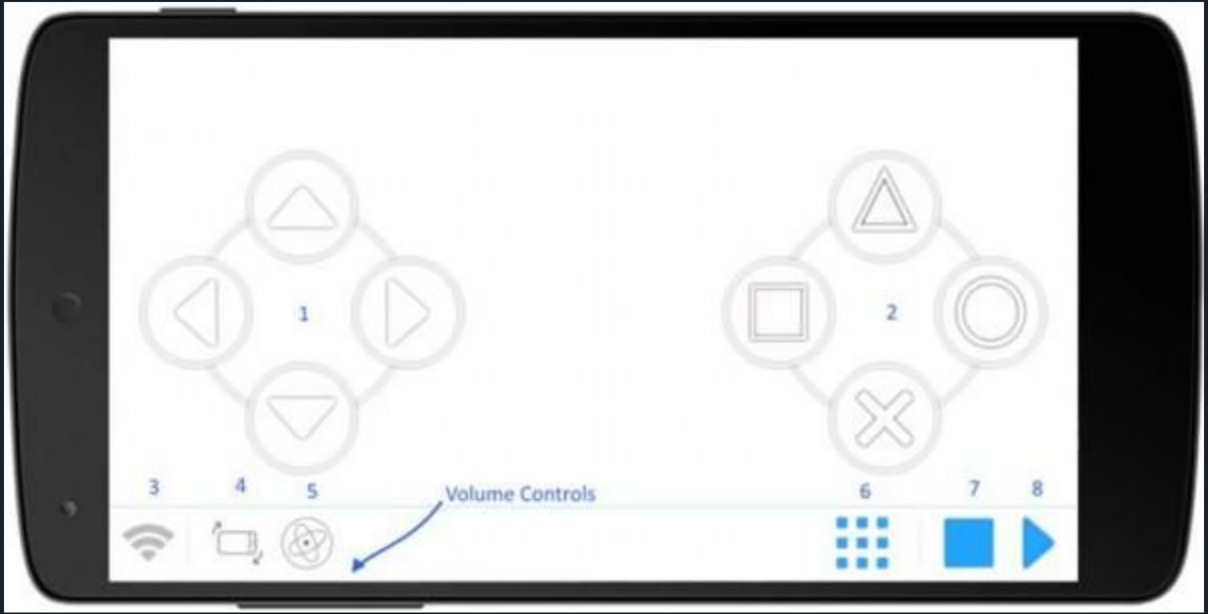
Long tap

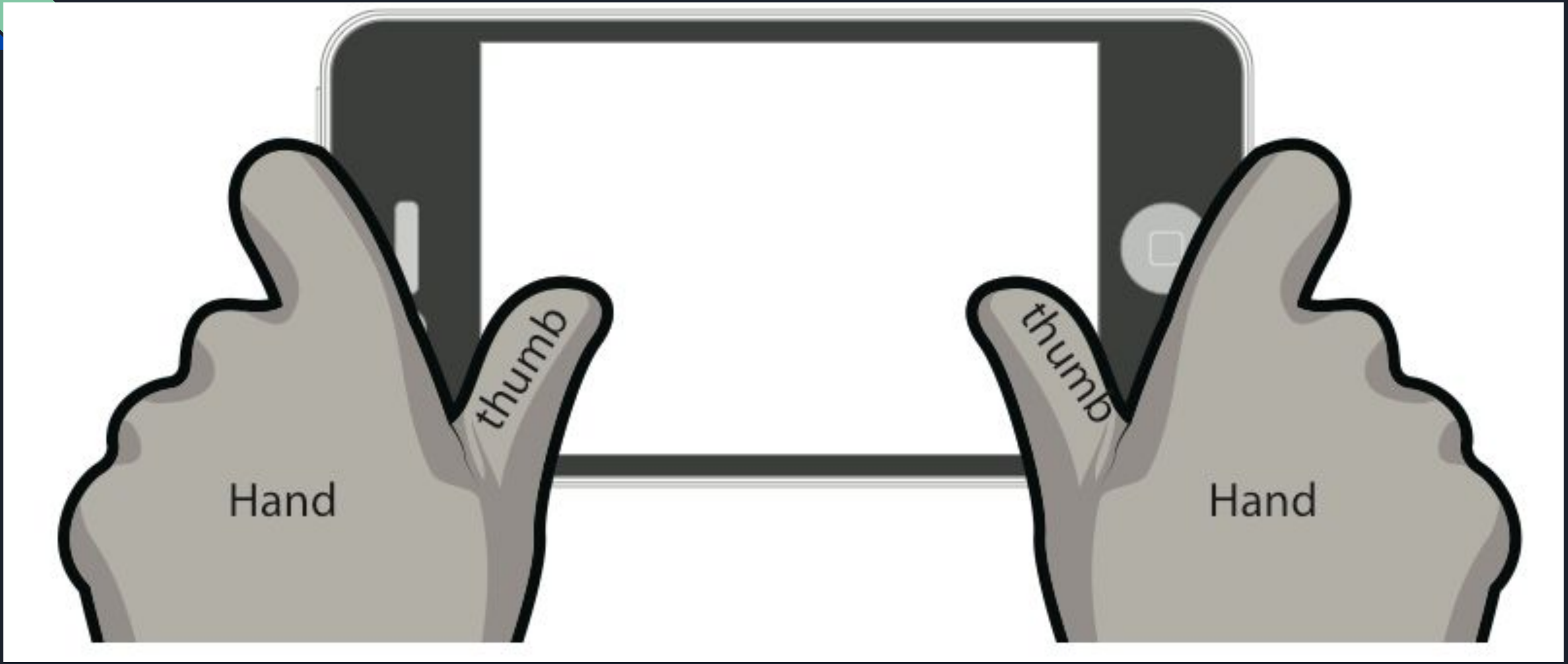
Slide

Gestures (pinch, pull, shapes, etc)

Orientation (Accelerometer)

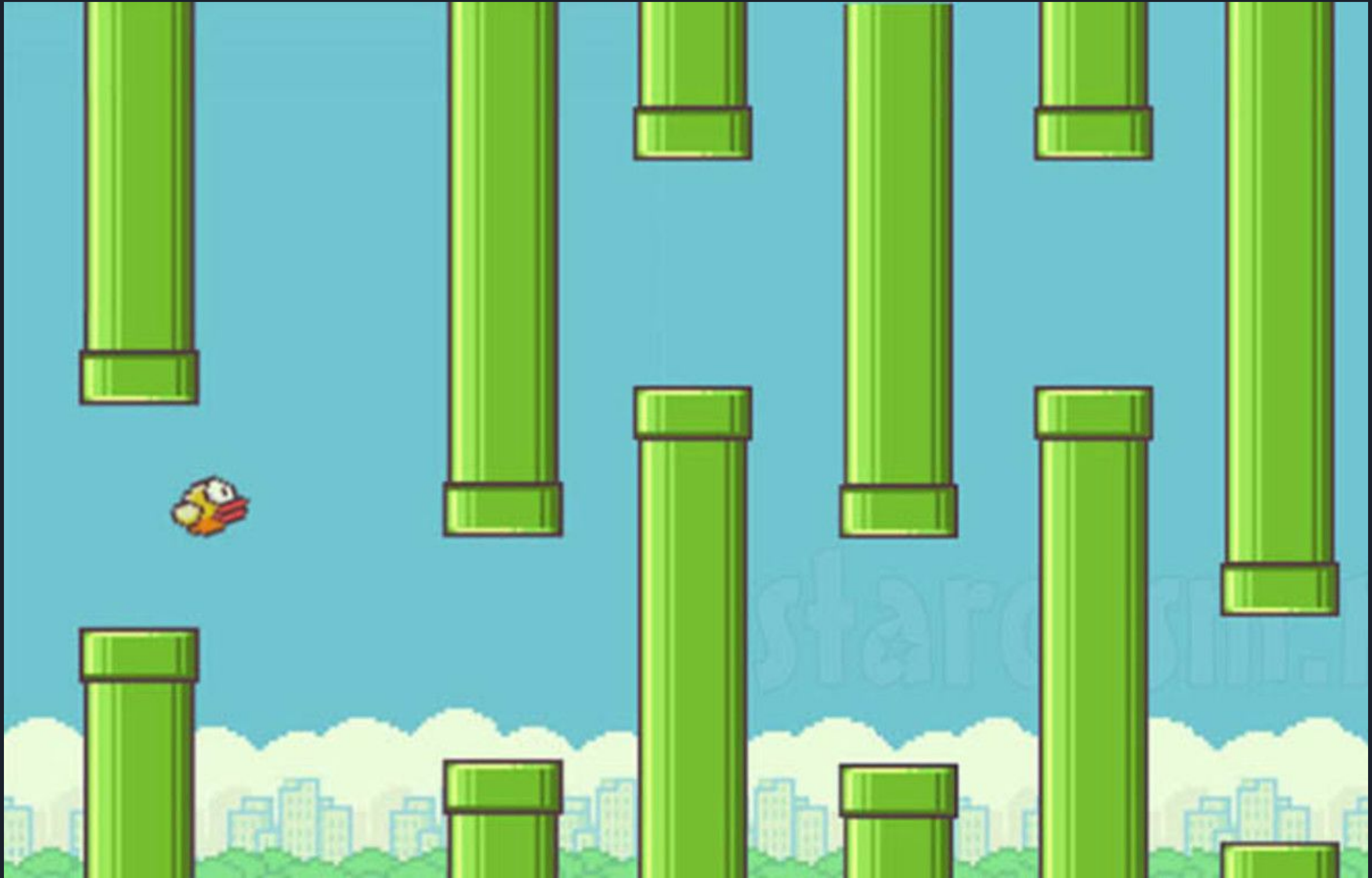
Hardware buttons


1
2
3
4
5
Volume Controls
6
7
8


thumb
Hand
thumb
Hand

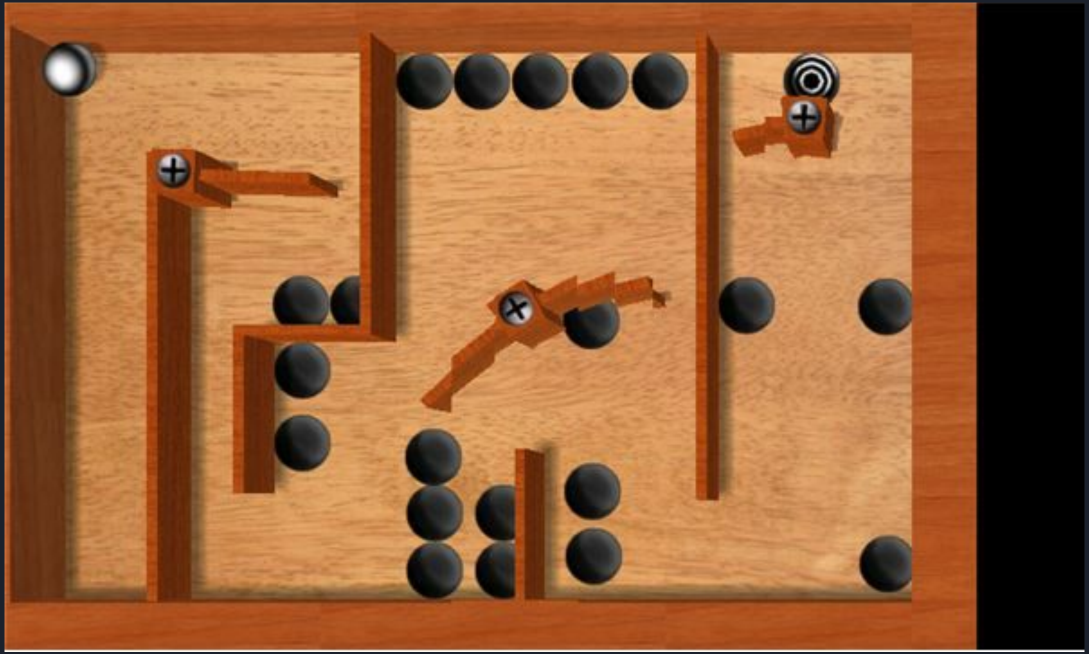
9TO5Google

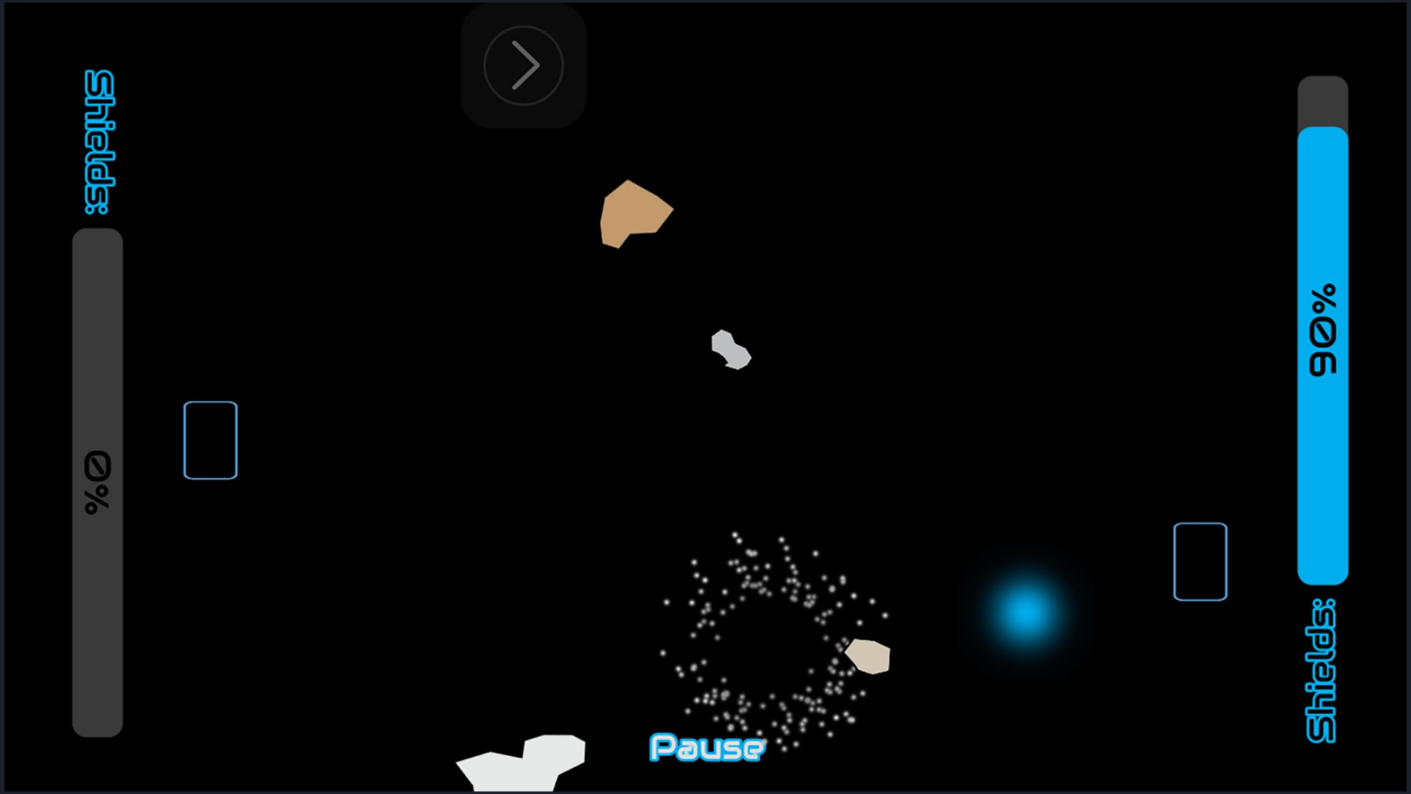
123
BEST: 189
7 FRUIT
COMBO
+7

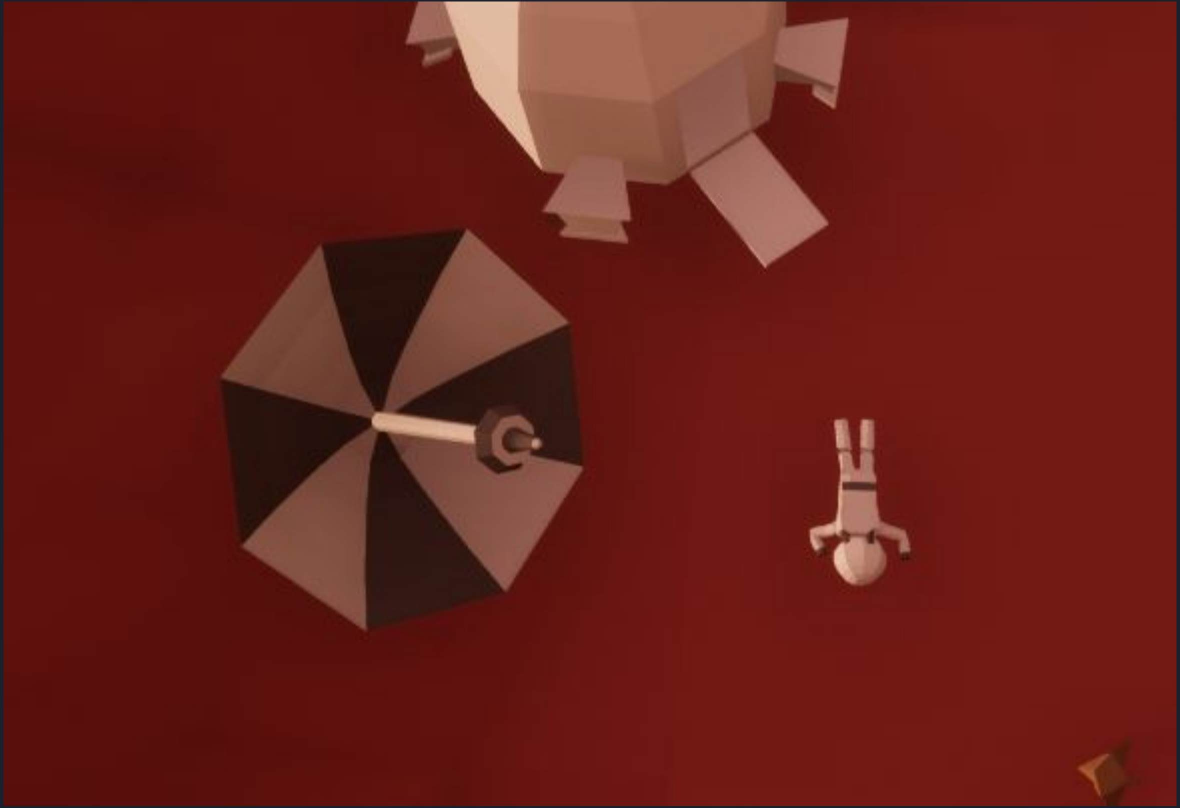


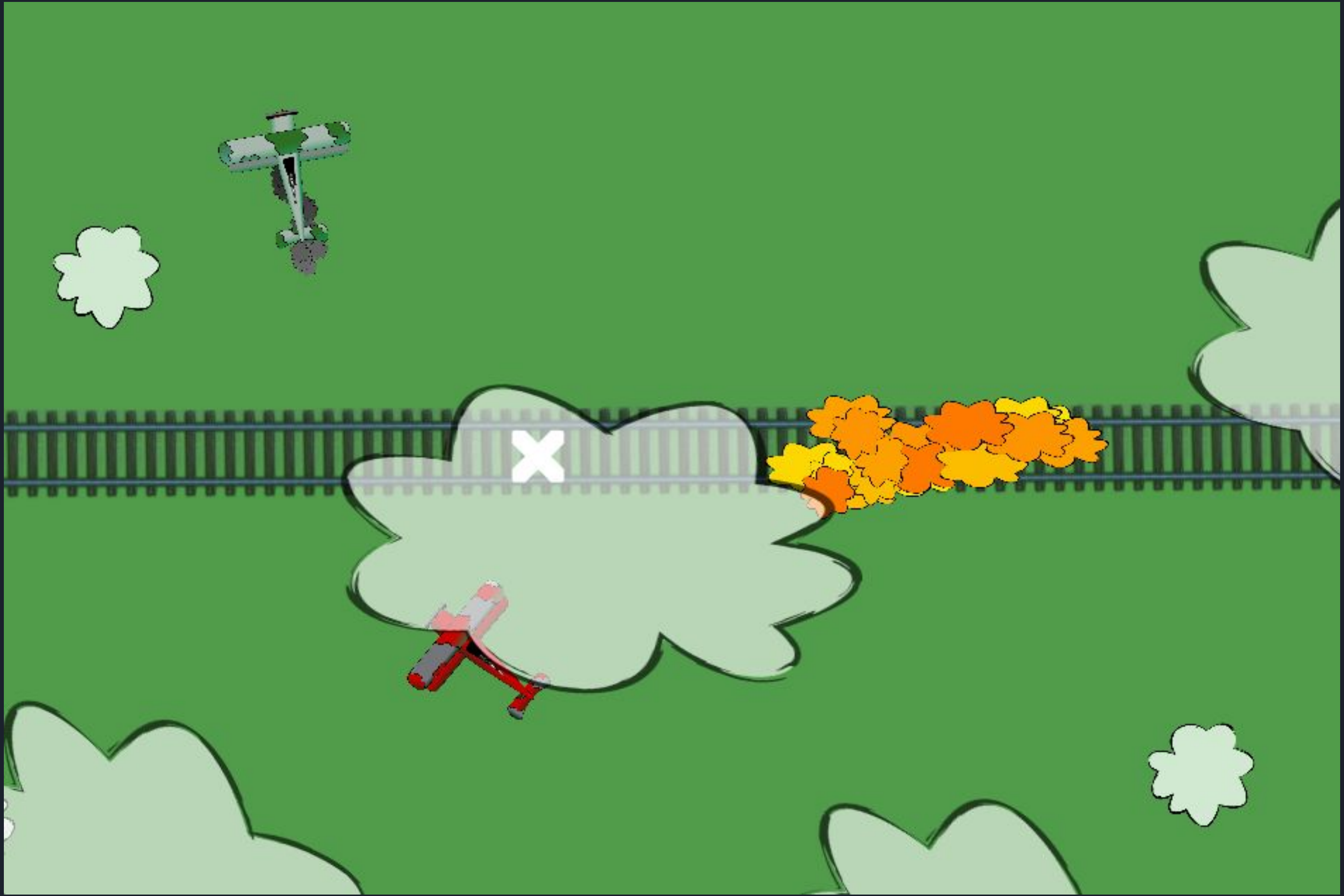
K LINE
D

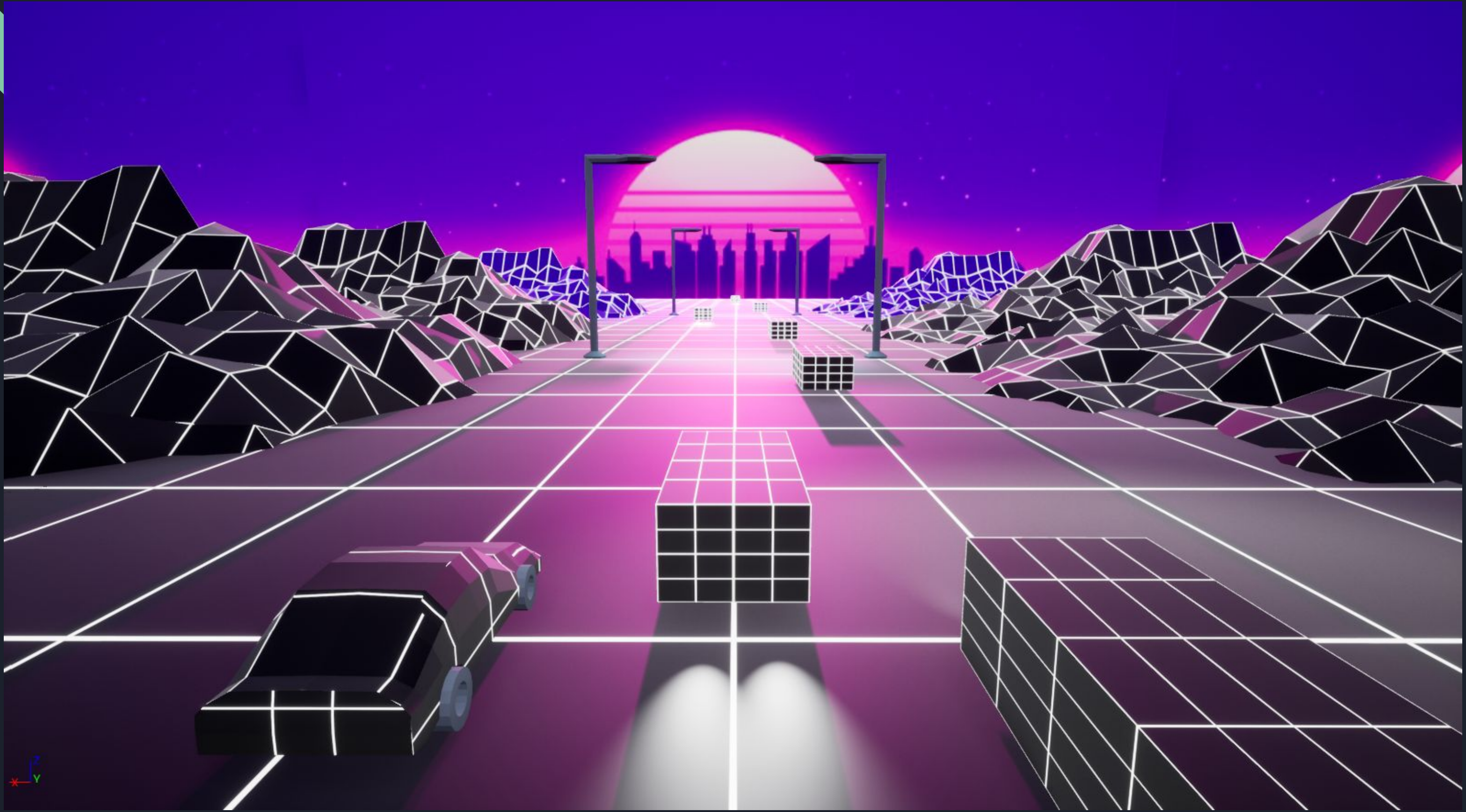
Time: 40
CrystalGeyserASW.com
Score: 0
CRYSTAL GEYSER
NATURAL
ALPINE SPRING WATER
BY CG ROXANE

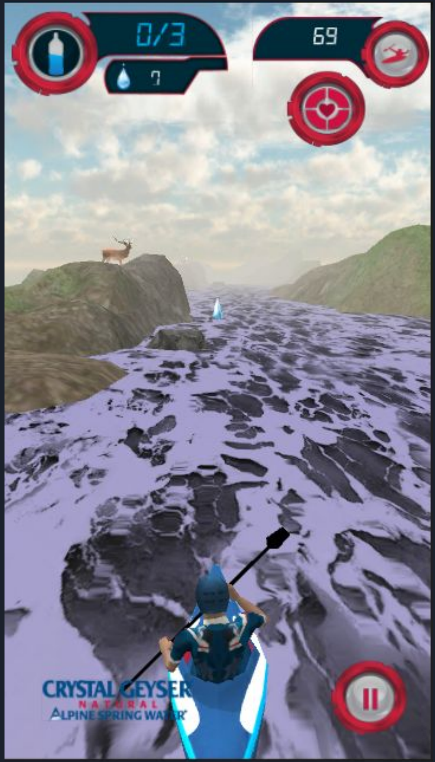
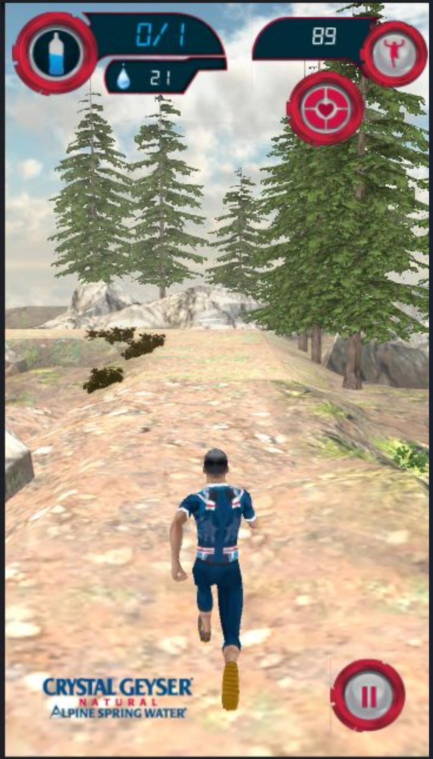


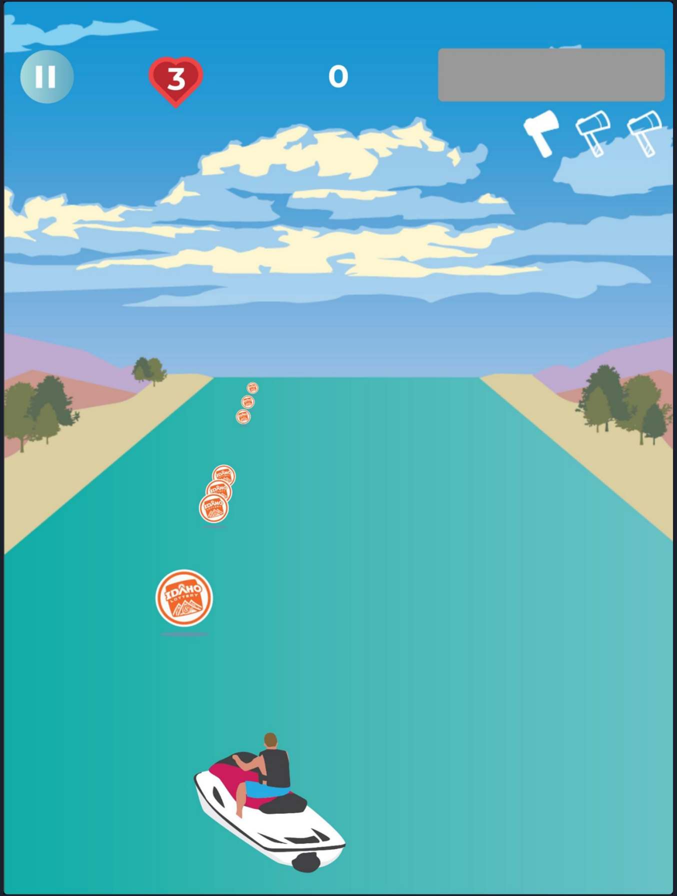

Shields:
0%
90%
Shields:
Pause



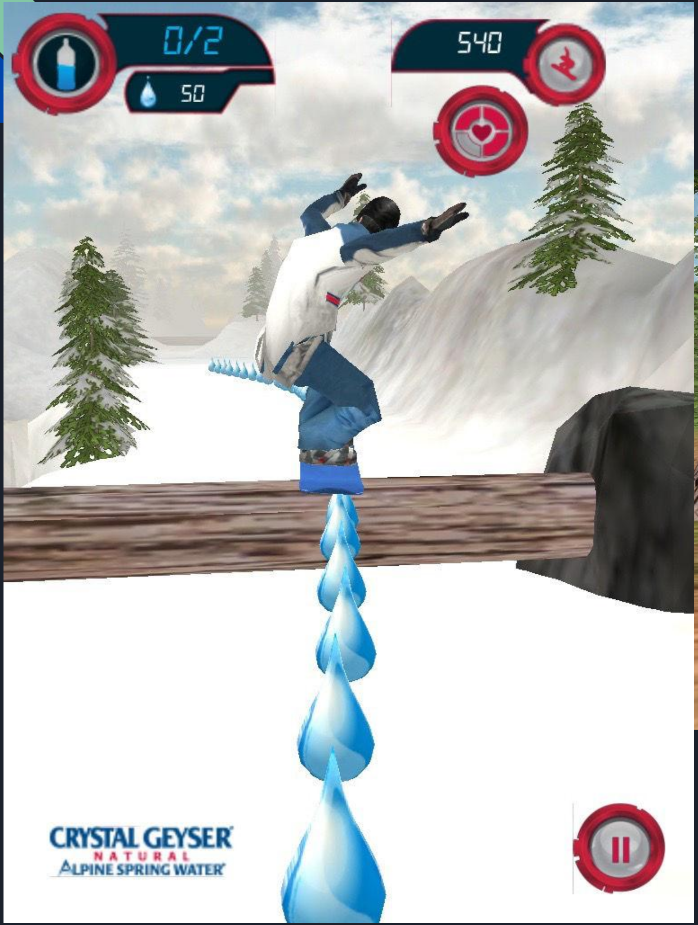



Z
Y
X


0/3
7
69
CRYSTAL GEYSER
NATURAL
ALPINE SPRING WATER

0/1
21
89
CRYSTAL GEYSER
NATURAL
ALPINE SPRING WATER


3
0

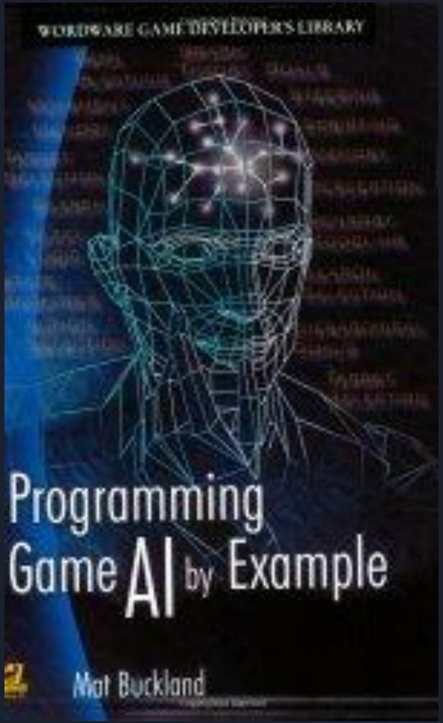
3


0/2
50
540
CRYSTAL GEYSER
NATURAL
ALPINE SPRING WATER

ALPINE SPRING WATER
CRYSTAL GEYSER

Artificial Intelligence

Programming Game AI by Example


WORDWARE GAME DEVELOPER'S LIBRARY
Programming Game AI by Example
Mat Buckland

Development SDKS

Pros and Cons

Lib GDX

Pros:

Written in Java

Windows, Mac, Linux, Android, iOS and HTML5

Low level

Heavy OpenGL Support

Open Source

Cons:

iOS support costs money

Low level

iOS emulator support

http://libgdx.badlogicgames.com/

Unity

Pros:

Awesome 2D/3D support

Pre-fabbed content for fast game development

Support for 1st or 3rd person

Asset store

Cons:

Costly ($399/year)

High learning curve with tools and APIs

Bugs at the top level

http://unity3d.com/

Unreal Engine

Pros:

Awesome 2D/3D support

Blueprints for visual programming

Support for 1st or 3rd person

Marketplace

Free (< $1 mil, then 5%)

Cons:

High learning curve with tools and APIs

C++ (or a plus) - but not needed

Huge install and overhead

http://unrealengine.com

Cocos 2d

Pros:

Large developer community

Free

Open Source

C++, Lua and Javascript

Cons:

C++

May be complex for some beginners

http://www.cocos2d-x.org/

Solar 2D (Formerly Corona SDK)

Pros:

Easy to display and animate graphics

Tap listeners are easy to implement

Code compiles to native language

Native hardware support

Large developer community

Simulator Support

Cons:

Need to learn Lua

https://solar2d.com/

Godot Engine (gu-doh)

Pros:

Easy to display and animate graphics

Tap listeners are easy to implement

Code compiles to native libraries (C++ or WASM)

Native hardware support

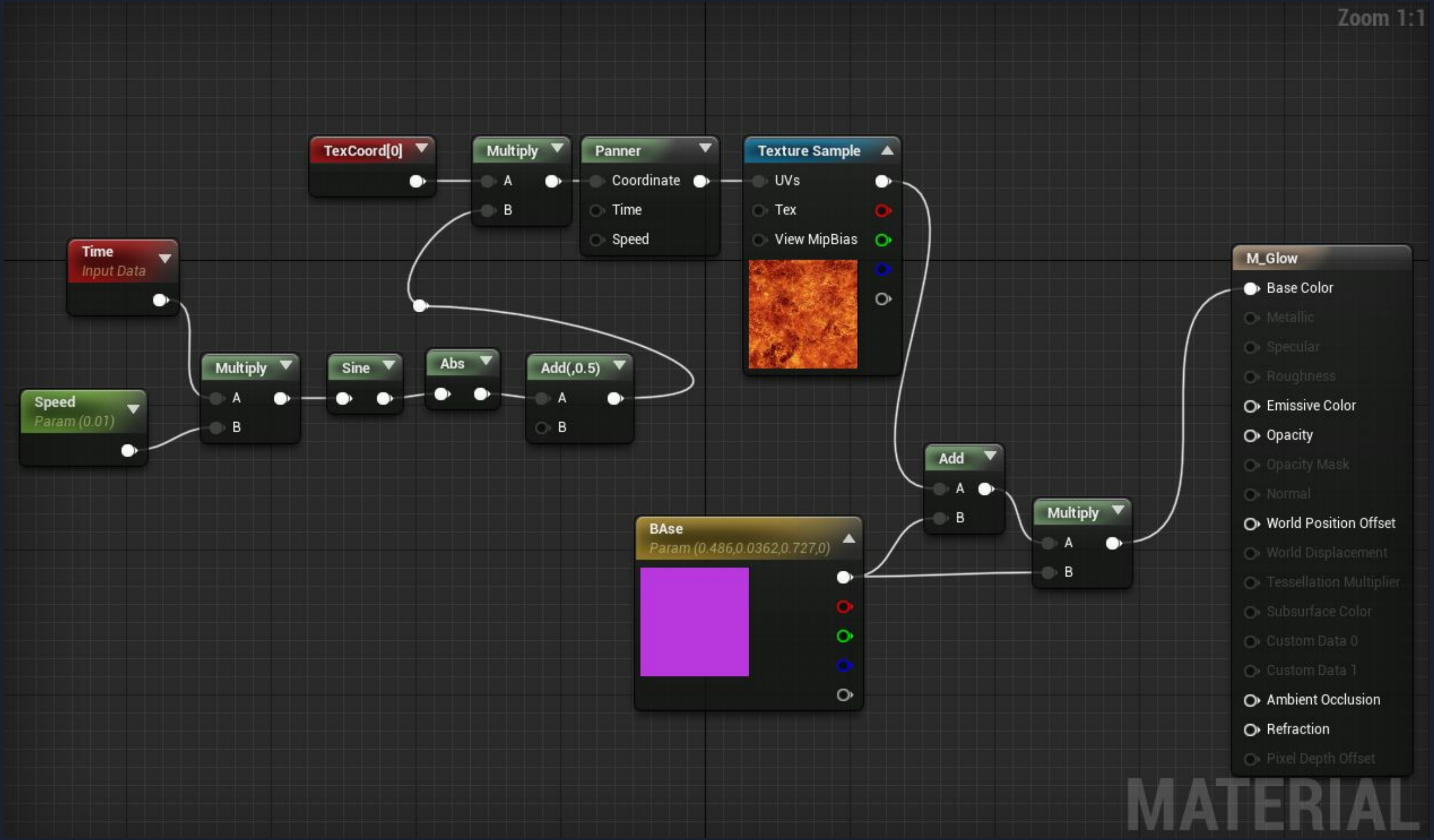
Large developer community

Physical Device Support (Android, Web, Desktop)

Cons:

Not every native mobile feature is supported and require plugins

http://godotengine.org/


Zoom 1:1
TexCoord[0]
Multiply
A
B
Panner
Coordinate
Time
Speed
Texture Sample
UVs
Tex
View MipBias
Time
Input Data
Speed
Param (0.01)
Multiply
A
B
Sine
Abs
Add(,0.5)
A
B
Add
A
B
BAse
Param (0.486,0.0362,0.727,0)
Multiply
A
B
M_Glow
Base Color
Metallic
Specular
Roughness
Emissive Color
Opacity
Opacity Mask
Normal
World Position Offset
World Displacement
Tessellation Multiplier
Subsurface Color
Custom Data 0
Custom Data 1
Ambient Occlusion
Refraction
Pixel Depth Offset
MATERIAL

3D Game Terminology

Meshes: Your 3D object’s geometry

Material: Skin or texture of your mesh

Shader: Algorithm to take an incoming photon and how that reflects, scatters or gets absorbed.

3D Game Terminology

Unity Prefab: Combination of meshes, materials, shaders, animations, scripts etc to allow the developer to drag and drop these prefabricated objects into the scene and have them just work.

Unreal Blueprints