

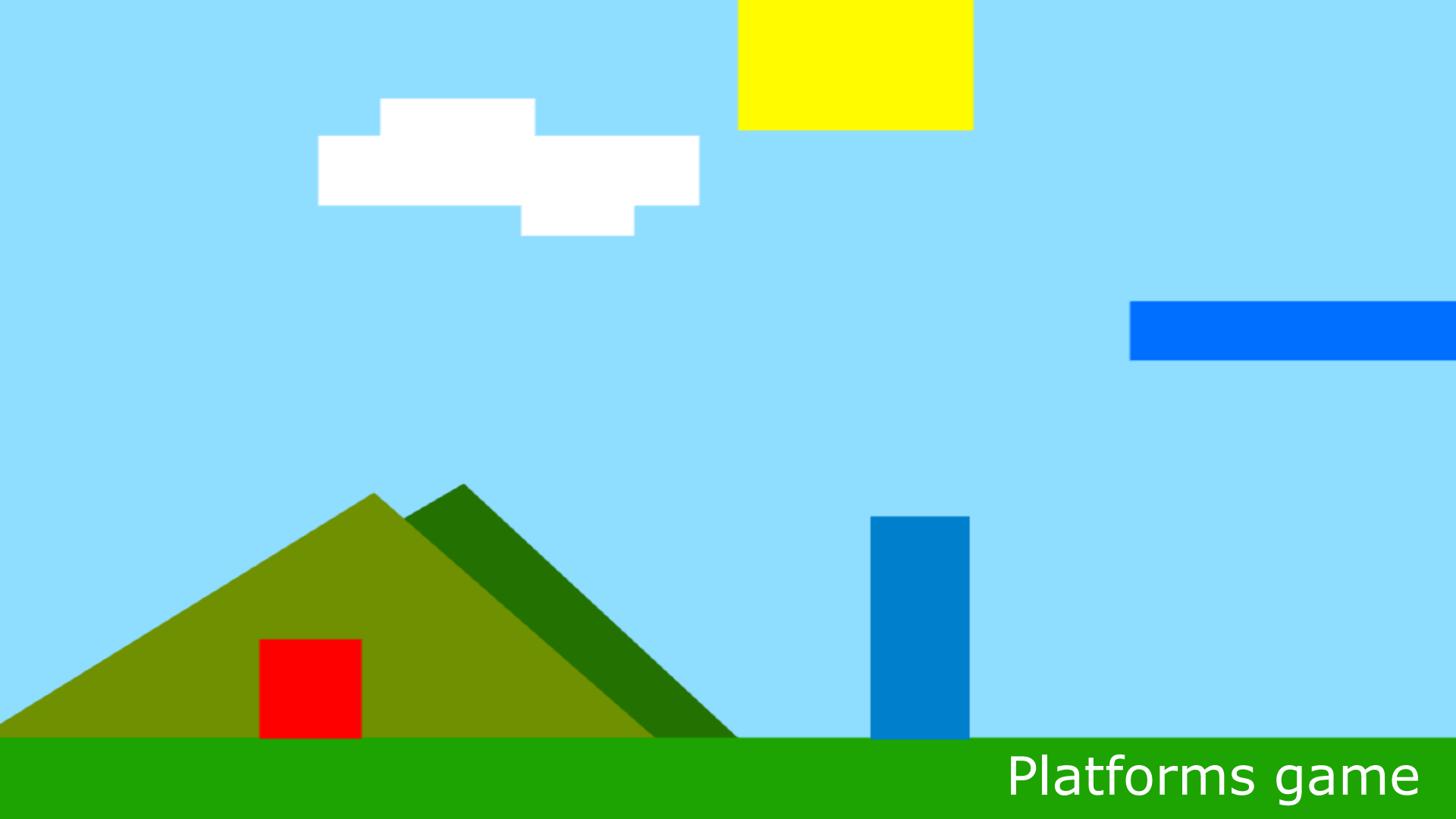
My Adventure

Tiago Costa | Rui Pereira | Miguel Moreira

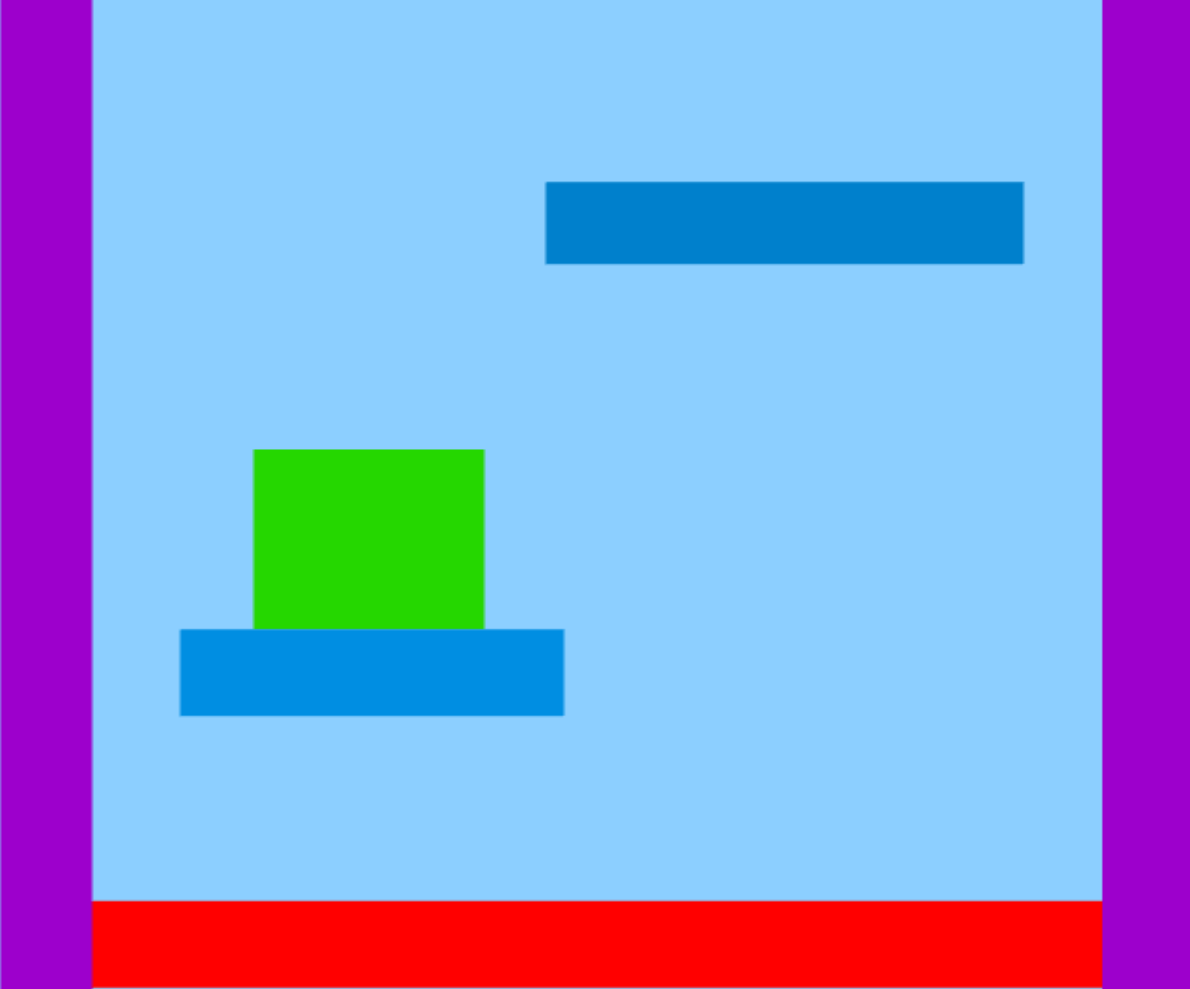
Sistemas Multimédia - 11/Jun/2014

Overview

- 2D game development tool
- Drag-and-drop actor placement
- Visual programming language to create gameplay mechanics



Platforms game



Platforms game

State of the Art



- C#
- No level editor

A screenshot of the Visual Studio IDE showing a C# project for XNA. The top status bar indicates the process is 'iexplore.exe' and the thread is '[3532] Main Thread'. The file explorer shows 'app.css', 'index.html', and 'app.ts'. The Solution Explorer shows a 'SimpleGame (class)' project. The Code Explorer shows the 'SimpleGame' class with methods 'constructor()', 'preload()', and 'create()'. A context menu is open over the 'SimpleGame' class, listing various properties and methods such as '_proto_', '_codePaused', '_paused', 'add', 'antialias', 'cache', 'camera', 'canvas', 'config', 'context', 'debug', 'device', 'height', and 'id'. The code in the background shows the 'create()' method with a 'text' variable and a 'style' object.

```
class SimpleGame {
    constructor() {
        this.game = new Phaser.Game(800, 600, Phaser.AUTO, 'content', { preload: thi
    }

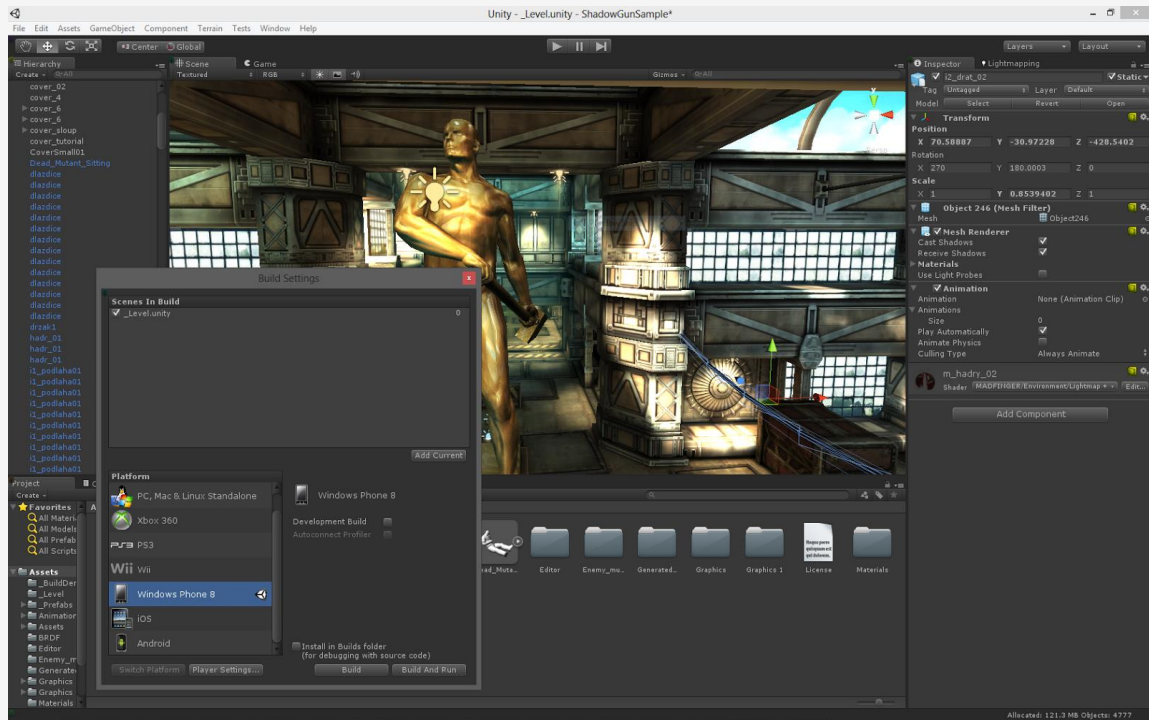
    game: Phaser.Game

    preload() {
        this.game.preload();
    }

    create() {
        var text = new Phaser.Game.Text(100, 100, 'Hello World', { fill: "#ff0044", align: "center" });
        this.game.add.sprite(100, 100, text);
    }
}

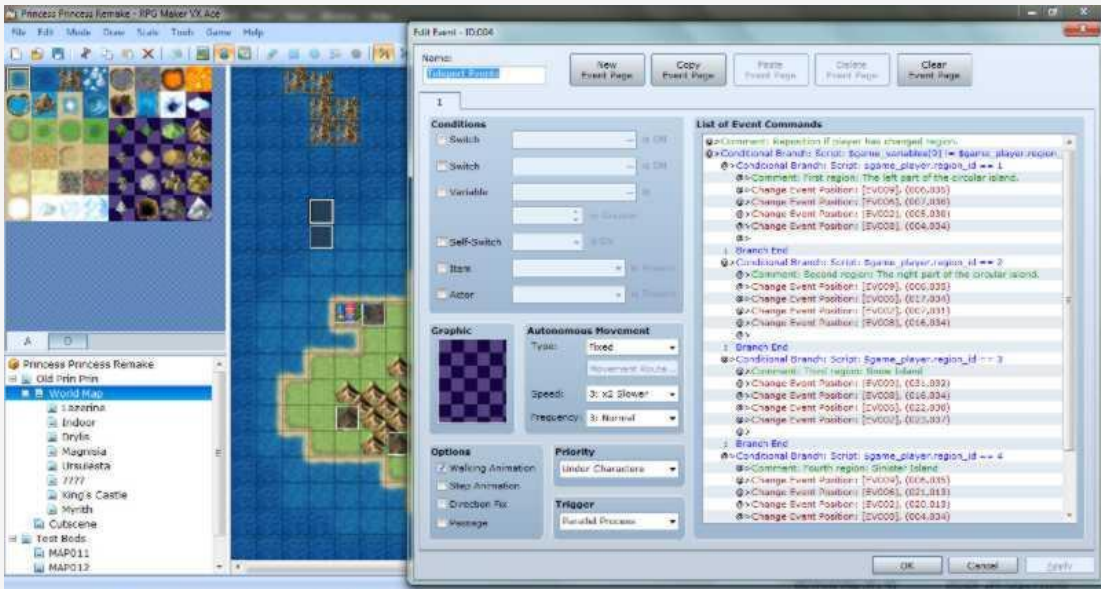
window.onload = () => {
    var game = new SimpleGame();
}
```

State of the Art

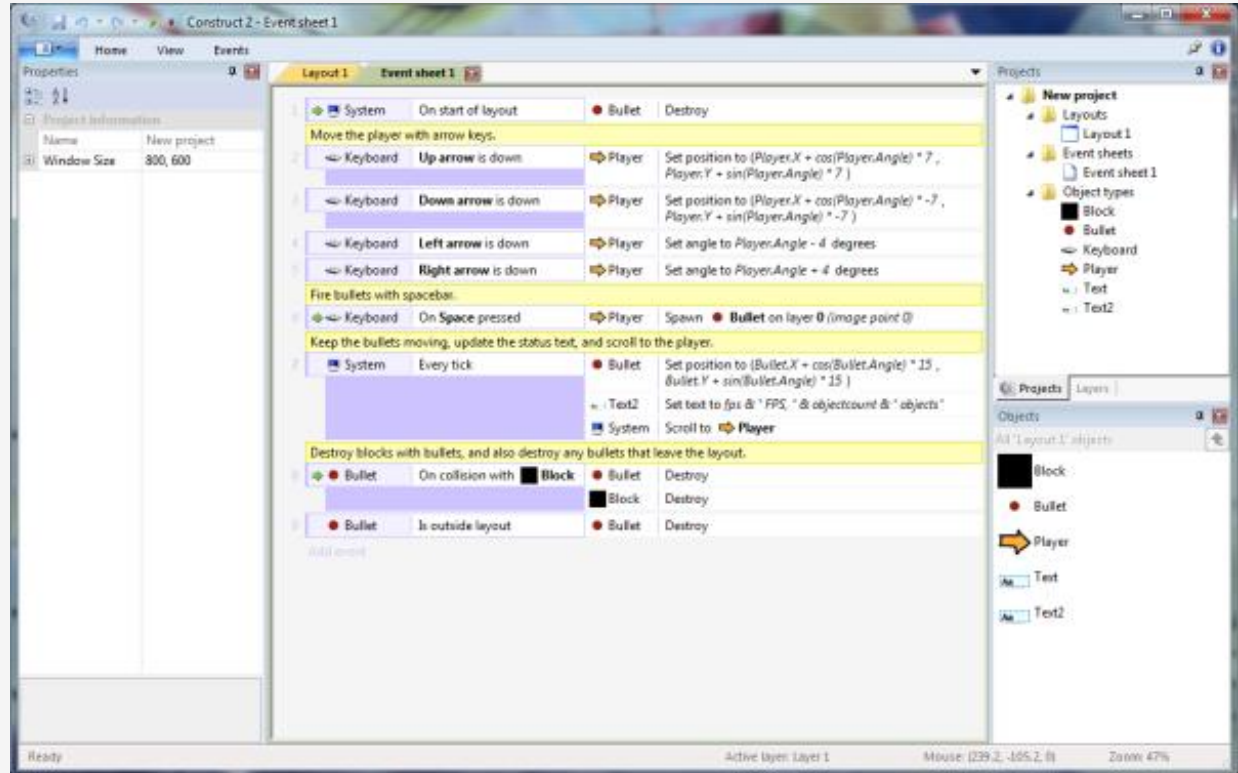


State of the Art

RPG MAKER



State of the Art



State of the Art

LittleBigPlanet



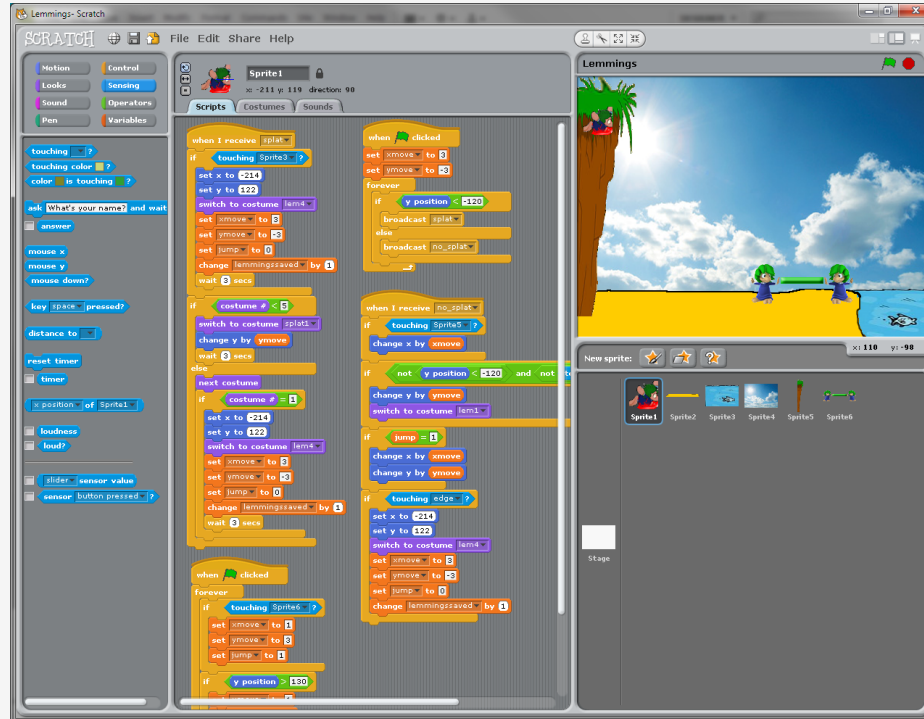
State of the Art

Kodu



State of the Art

Scratch

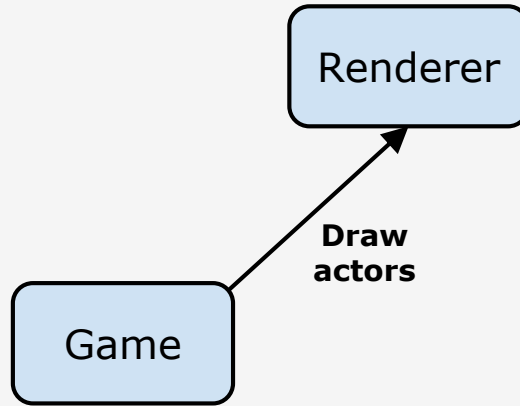


Challenge

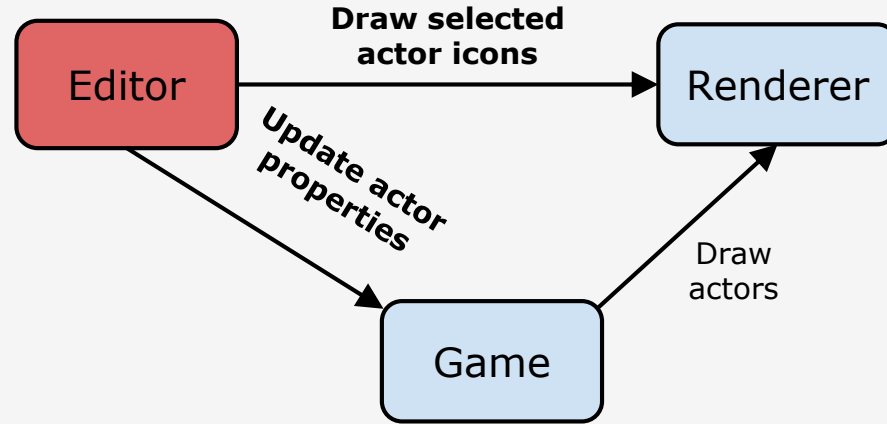
Create a tool to:

- develop 2D games in under 10 minutes
- no programming experience required

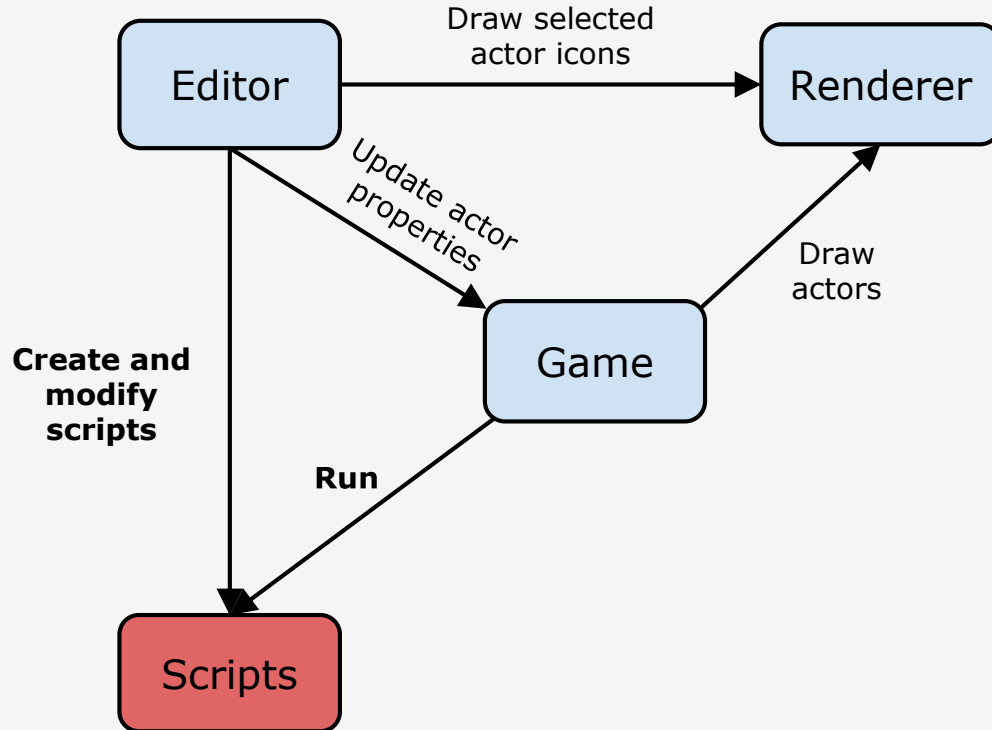
Tool Architecture



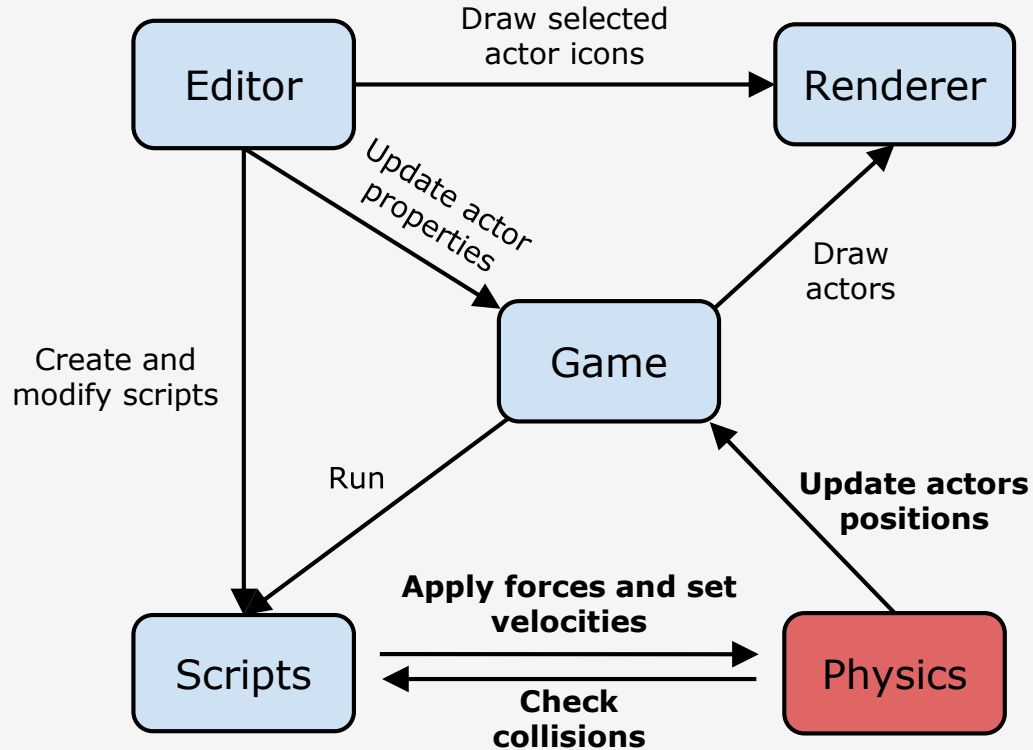
Tool Architecture



Tool Architecture



Tool Architecture



Game

Actors:

- Name
- Class - identify collisions between actors
- Color
- Type - background, dynamic, kinematic or static
- Script - defines the behaviour of this actor

Level Editor

Game customization Actor customization Scripts

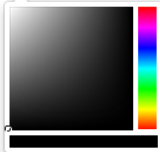
Start

Download game

Choose File No file chosen

Player: -- Choose actor -- ▾

Background color:



Gravity:

View distance:

Level Editor

Game customization Actor customization Scripts

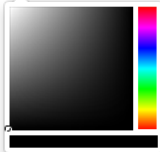
Start

Download game

Choose File No file chosen

Player: -- Choose actor -- ▾

Background color:



Gravity:

View distance:

Level Editor

Game customization Actor customization Scripts

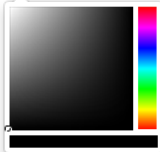
Start

Download game

Choose File No file chosen

Player: -- Choose actor -- ▾

Background color:



Gravity:

View distance:

Level Editor

Game customization Actor customization Scripts

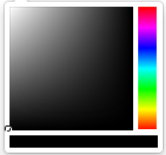
Start

Download game

Choose File No file chosen

Player: -- Choose actor -- ▾

Background color:



Gravity:

View distance:

Level Editor

Game customization Actor customization Scripts

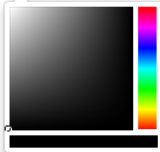
Start

Download game

Choose File No file chosen

Player: -- Choose actor -- ▾

Background color:



Gravity:

View distance:

Level Editor

Game customization Actor customization Scripts

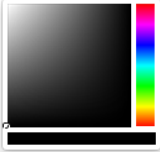
Start

Download game

Choose File No file chosen

Player: -- Choose actor -- ▾

Background color:



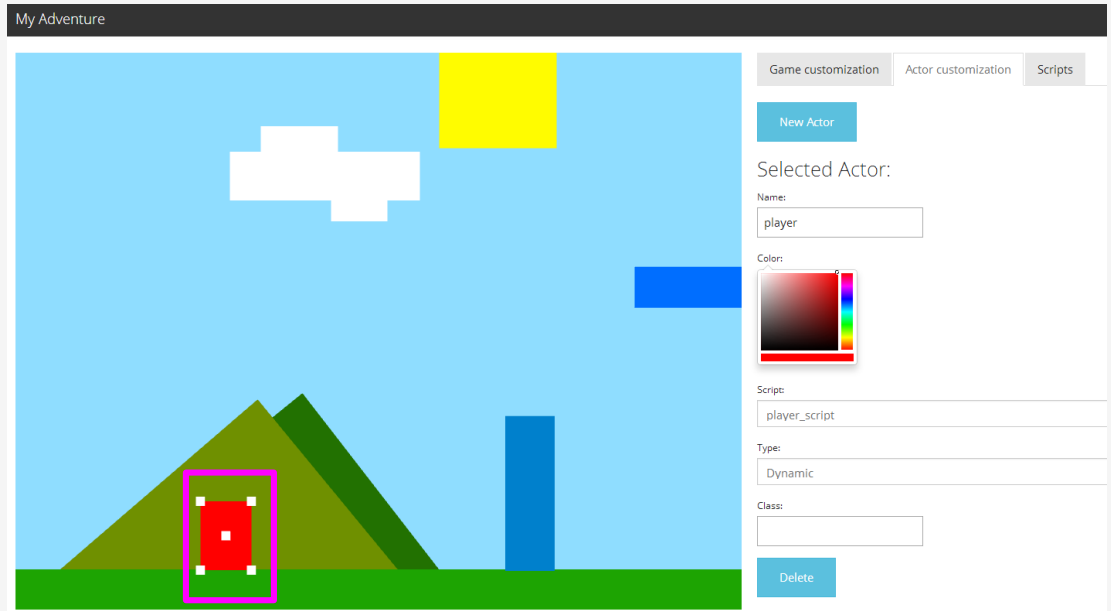
Gravity:

View distance:

Level Editor

- Drag-and-drop:

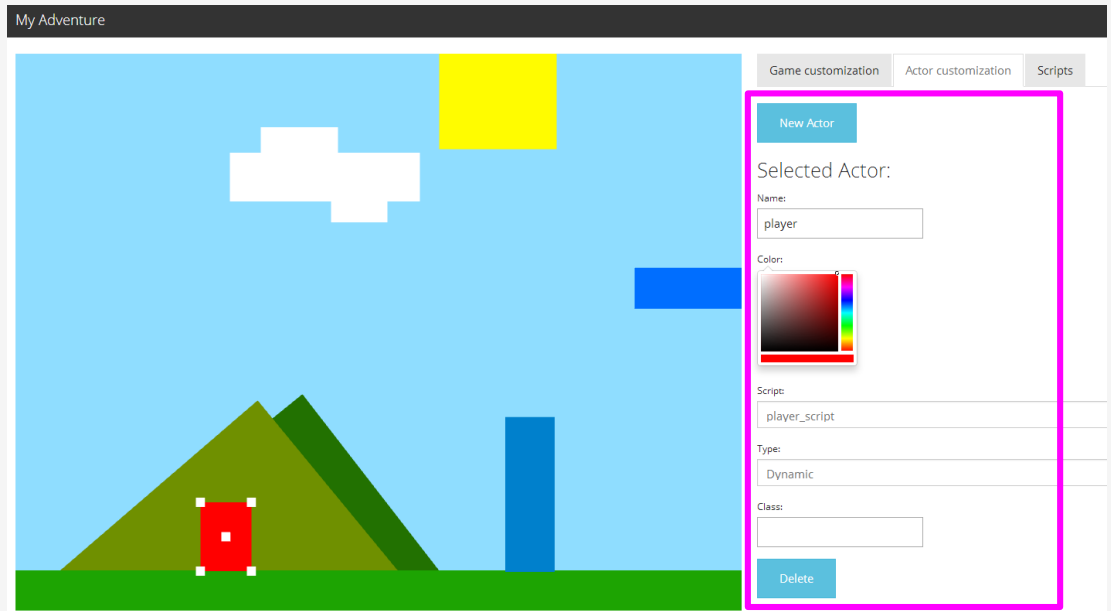
- Position
- Scale
- Rotate



Level Editor

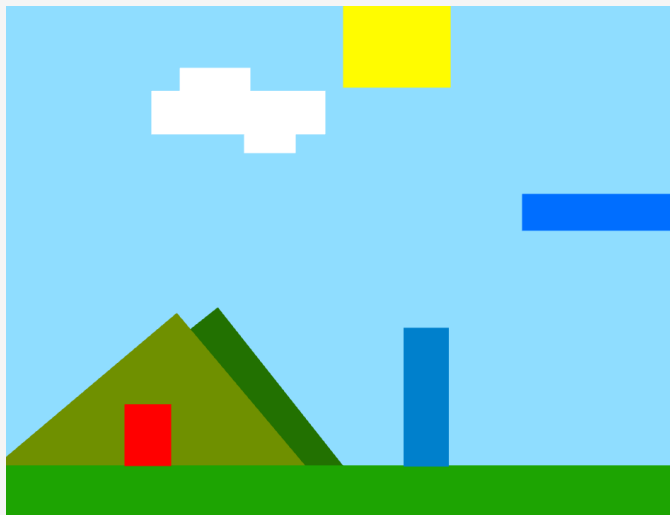
- Drag-and-drop:

- Position
- Scale
- Rotate

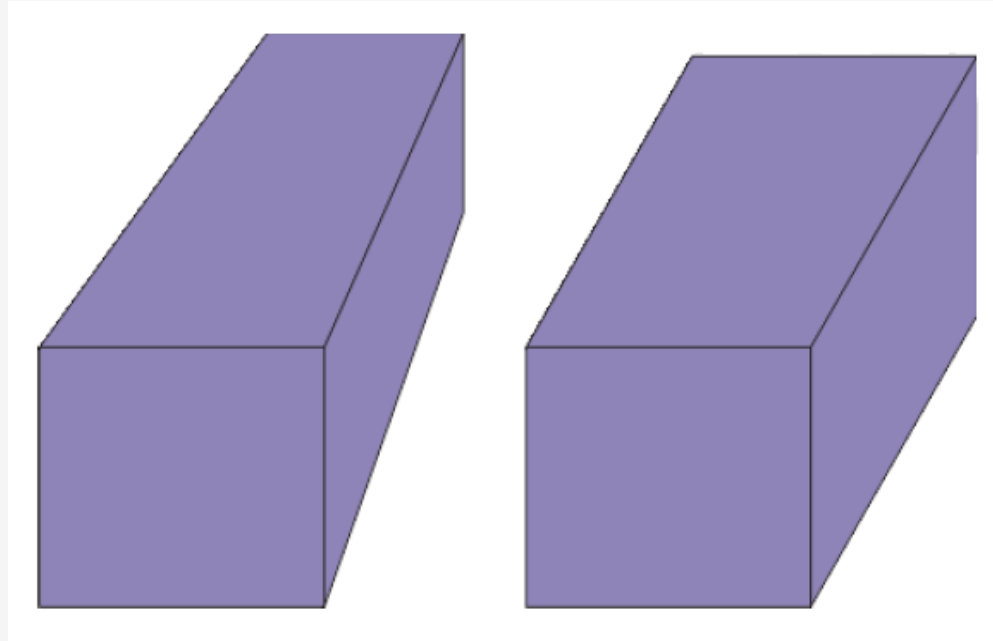


Renderer

- Orthographic projection
- Draw colored rectangles



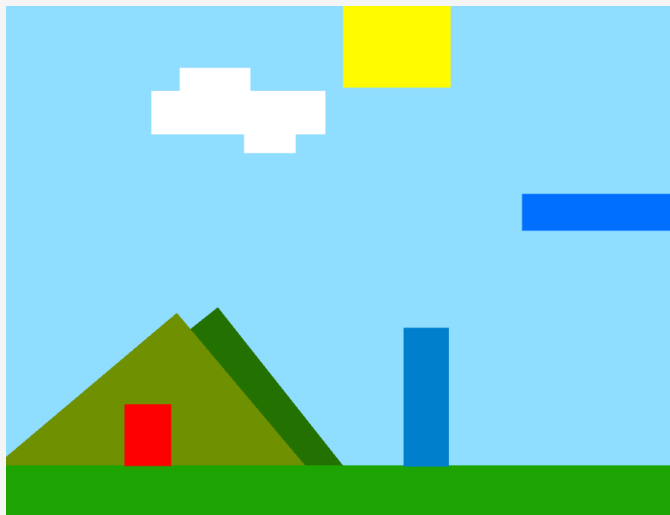
Projection



Perspective vs Orthographic projection

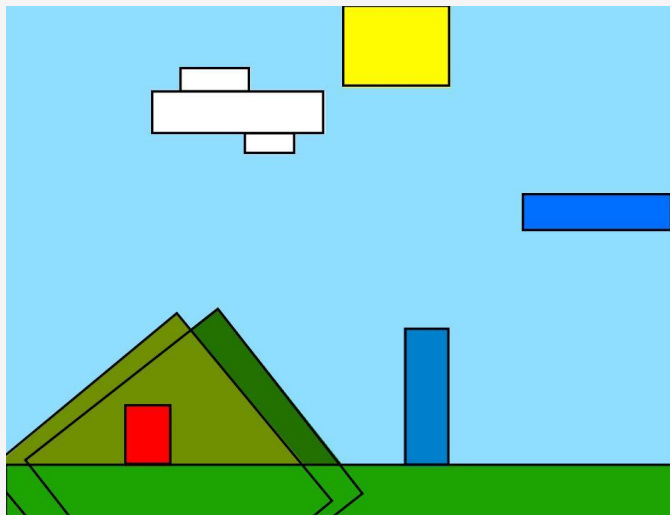
Renderer

- Orthographic projection
- Draw colored rectangles



Renderer

- Orthographic projection
- Draw colored rectangles



Physics

Game actors linked to physics bodies

Physics bodies can be:

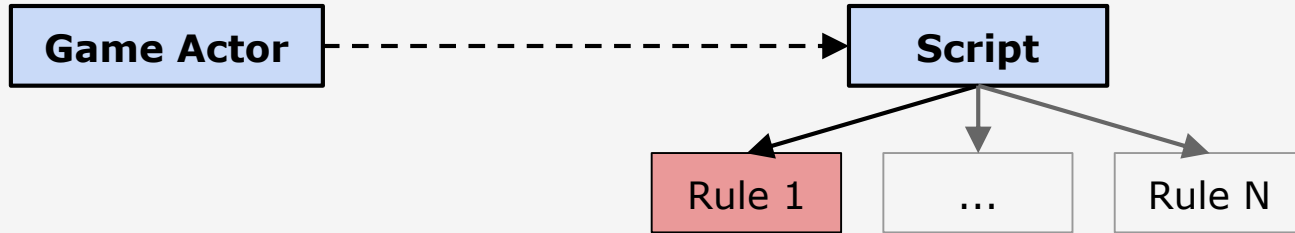
- Dynamic
 - Apply forces
- Kinematic
 - Set velocity
- Static



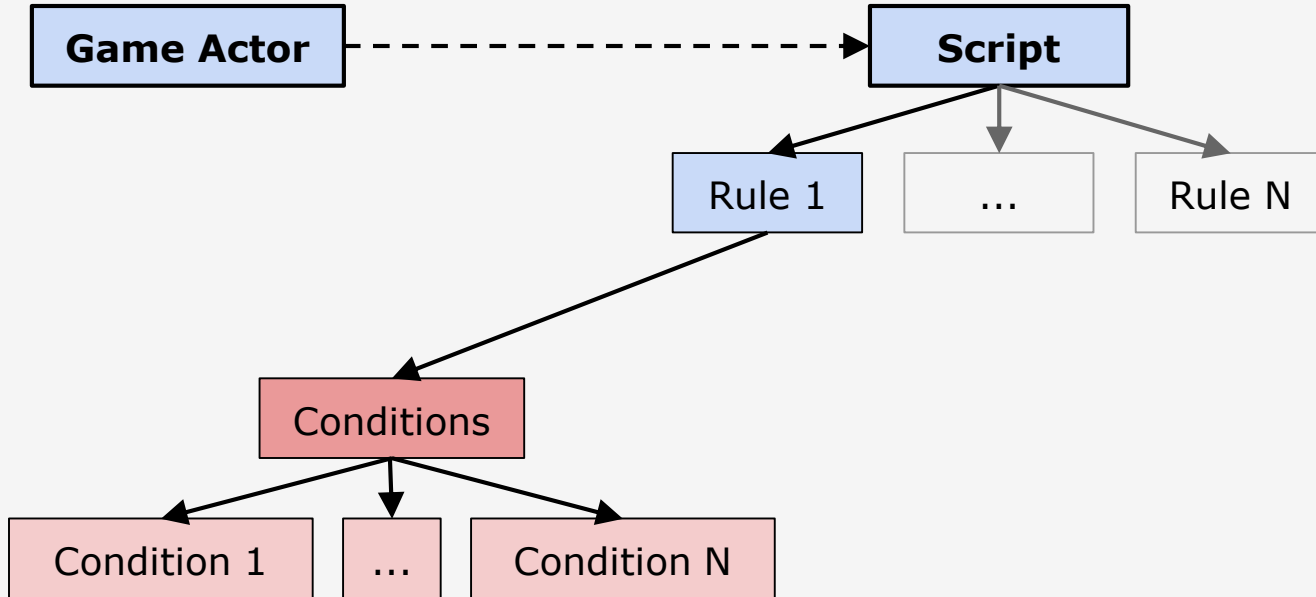
Scripting



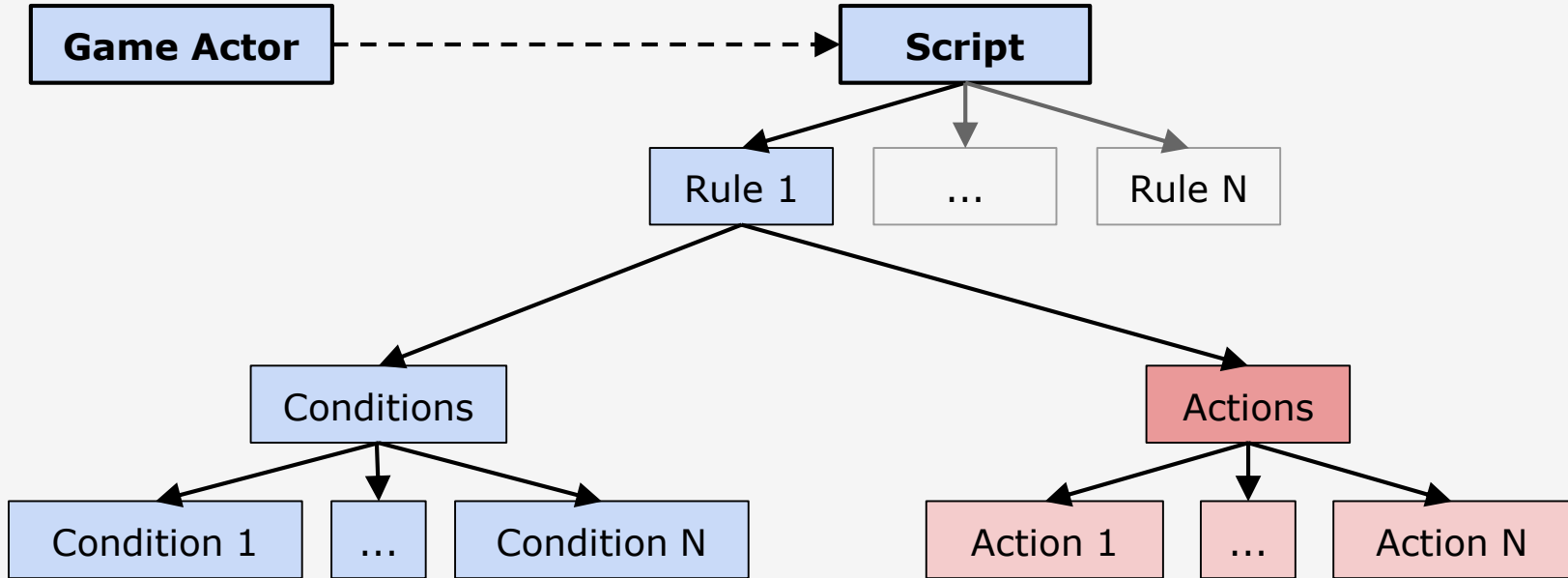
Scripting



Scripting



Scripting



Scripting

- Conditions
 - If Key Pressed
 - If Collision (Actor class, Side)

Scripting

- Conditions

- If Key Pressed
- If Collision (Actor class, Side)

Can be inverted

- If NOT Key Pressed
- If NOT Collision

Scripting

- Conditions

- If Key Pressed
- If Collision (Actor class, Side)

Can be inverted

- If NOT Key Pressed
- If NOT Collision

- Actions

- Move to
- Apply force
- Set velocity (X/Y/Both)
- Destroy

Scripting - Example

Player movement:

- Rule 1 (move left):

Conditions:

- If Key pressed: LEFT

Actions:

- Set Velocity (X): -10

Scripting - Example

Player movement:

- Rule 1 (move left):

Conditions:

- If Key pressed: LEFT

Actions:

- Set Velocity (X): -10

- Rule 2 (move right):

Conditions:

- If Key pressed: RIGHT

Actions:

- Set Velocity (X): 10

Scripting - Example

- Rule 3 (stand still):

Conditions:

- If NOT Key pressed: LEFT
- If NOT Key pressed: RIGHT

Actions:

- Set Velocity (X): 0

Scripting - Example

- Rule 4 (jump):

Conditions:

- If Key pressed: UP
- If Collision: Side -> Bottom

Actions:

- Apply Force: (0,500)

Live Demo

Conclusions

- Develop games in under 10 minutes
- Visual programming language allows the implementation of different gameplay mechanics

Questions