

CS 176

Advanced Scripting

Tile-Based Collision

Antoine Abi Chacra
Elie Abi Chahine

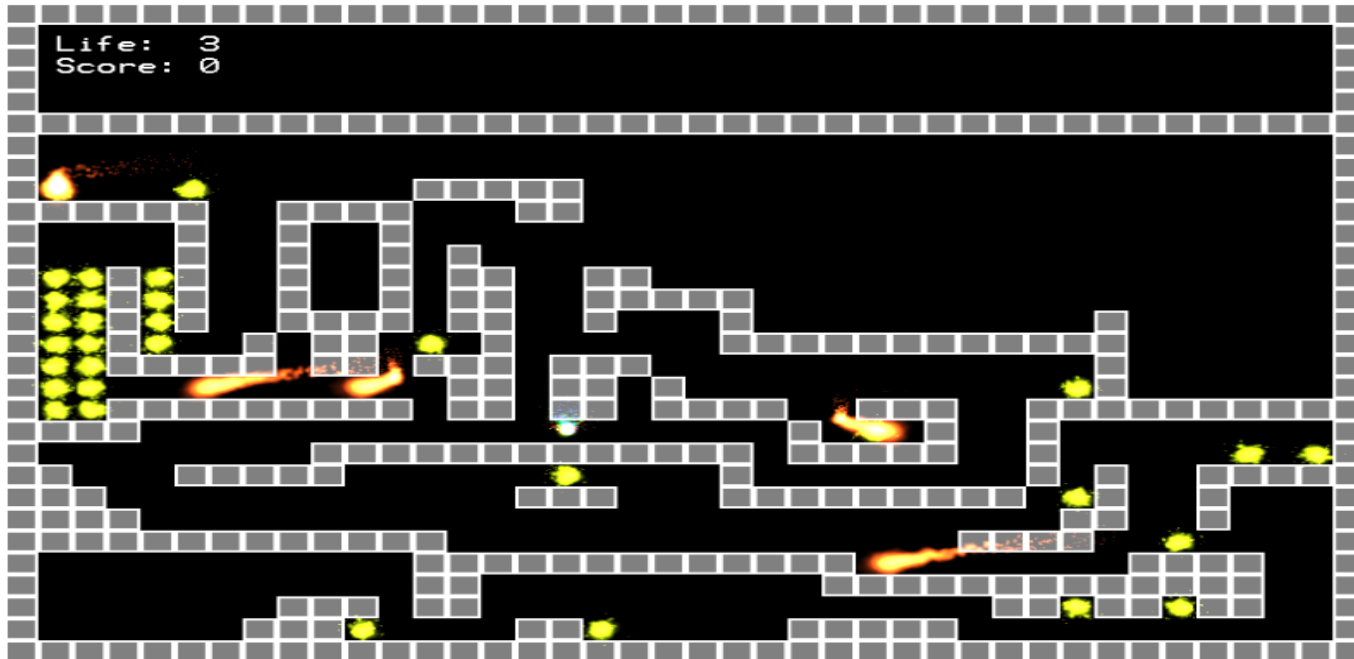
Binary Map

Introduction to binary maps

In games, players usually don't have a complete freedom while controlling the main character (or any controllable sprites). The movement is dictated by game play rules:

- In platform games, the main character can move horizontally on platforms, with the ability to jump a small distance vertically.
- In a “bricks” style game, the pad can only move horizontally.
- In maze-based games, where each level is divided into 2 areas, usable and unusable, game objects can only maneuver in “usable” areas.
- The same logic applies in 3D games, the player has always some kind of restrictions concerning moving and controlling game objects.

Introduction to binary maps



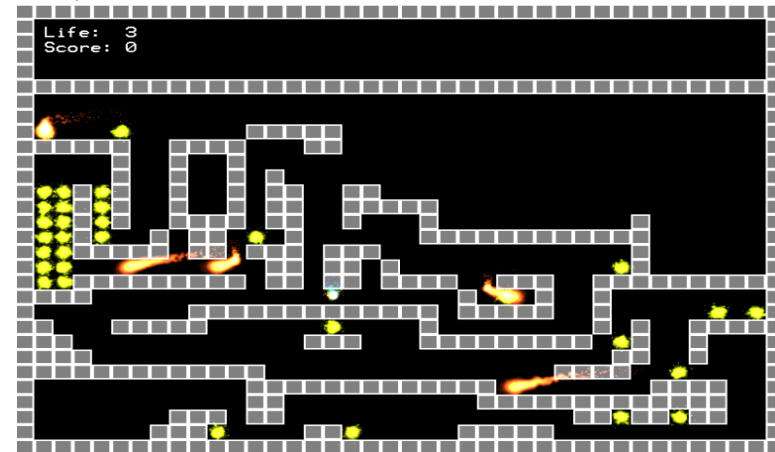
Example: Platform games.

- Movement of the main character is restricted: It can only walk on platforms
- Each level is divided into 2 main parts: Platforms and non-platforms (air).

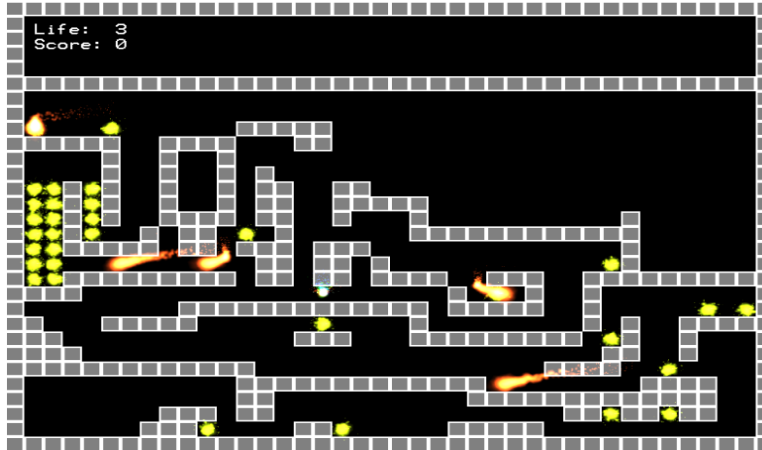
Introduction to binary maps

When the main character moves in any direction, we must check for collision against the platform walls.

- We can apply the traditional collision check, which uses the sprite's collision data
- Usually a bounding circle or rectangle
- This will force us to create a bounding rectangle on each platform and wall (in every single level) in order for the collision check to work.
 - Requires a lot of memory
 - Expensive computation since it will involve lots of collision checks between game objects and all the potential walls.



Introduction to binary maps



Instead of applying a traditional collision check for this case, we can use a binary map.

- It consists of dividing the level into a grid.
- Each cell inside this grid can hold 1 of 2 values: 0 or 1
- If the value of a certain cell is “0”, then it is considered as a non collision area.
- On the other hand, if its value is “1”, then it is considered as a collision area.

Binary map collision

- Binary map collision relies on the fact that the game world is a grid.
- Game object instances are able to “access” a new cell depending on that cell's value.
- Therefore, using a 2 dimensional array of “bools” to represent this 2D grid is perfectly suited for this problem, since each element in the 2D array will represent 1 cell (Tile on the screen).

Note: We will have a slightly enhanced version of Binary Map Collision in our engine.

Collision with the background Array

- The values in this array are used for 2 purposes:
 - To generate the background in a very fast way (which we already mentioned)
 - To create a collision array for the game object to check with

```

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1
1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

```

0 Non collision area

1 Collision area

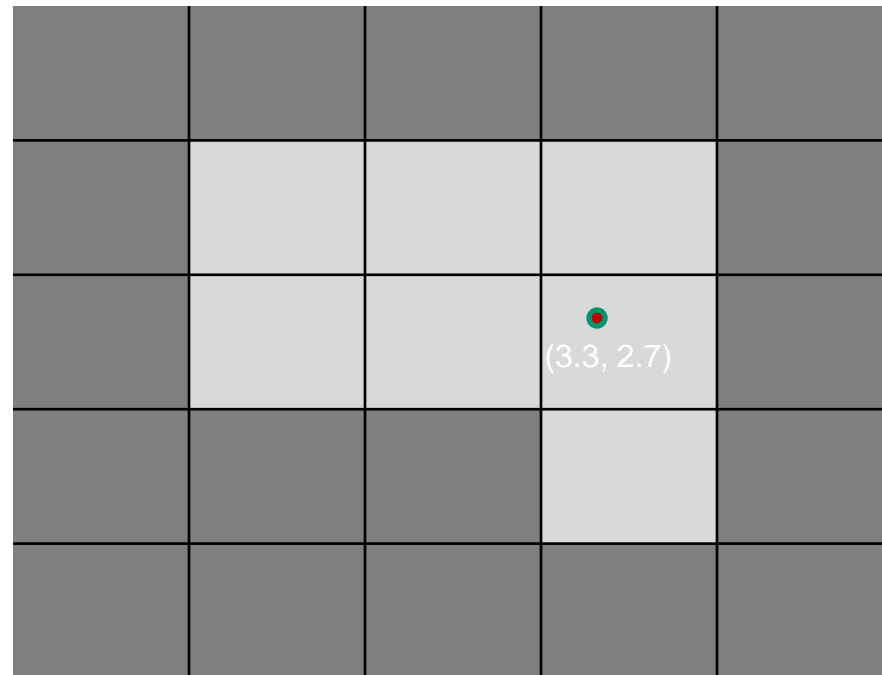
Point Collision

- Knowing the tile's dimension (width and height), to check if a point is in a "solid" cell we get its position in the array (using array indices) and check its value.

Collision Data

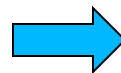
```

1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 1 1 0 1
1 1 1 1 1
  
```



The following is true if:

- Tile's (0,0) is the top left
- First tile starts at (0,0)
- All tiles and objects have positive X and Y values

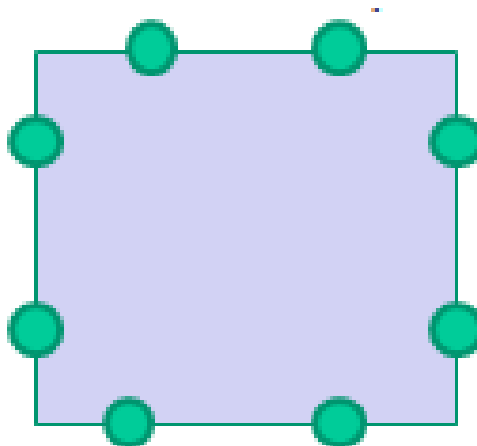


$$X = \text{ObjectX} / \text{TileWidth}$$

$$Y = \text{ObjectY} / \text{TileHeight}$$

Hot Spots

- Our object is not just one point but is encapsulated with a bounding rectangle
- We are dealing with more than one point
- These points are called “Hot Spots”

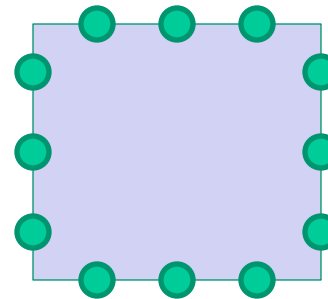
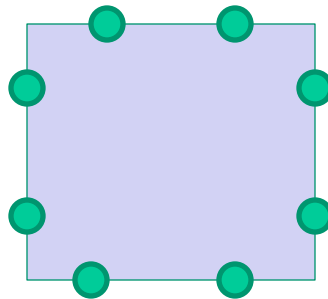
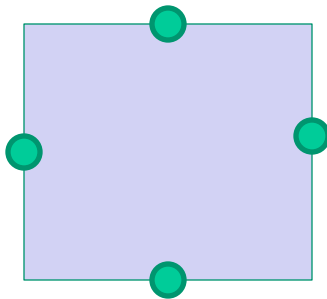


Hot Spots

Better Collision Accuracy



Faster Collision Check

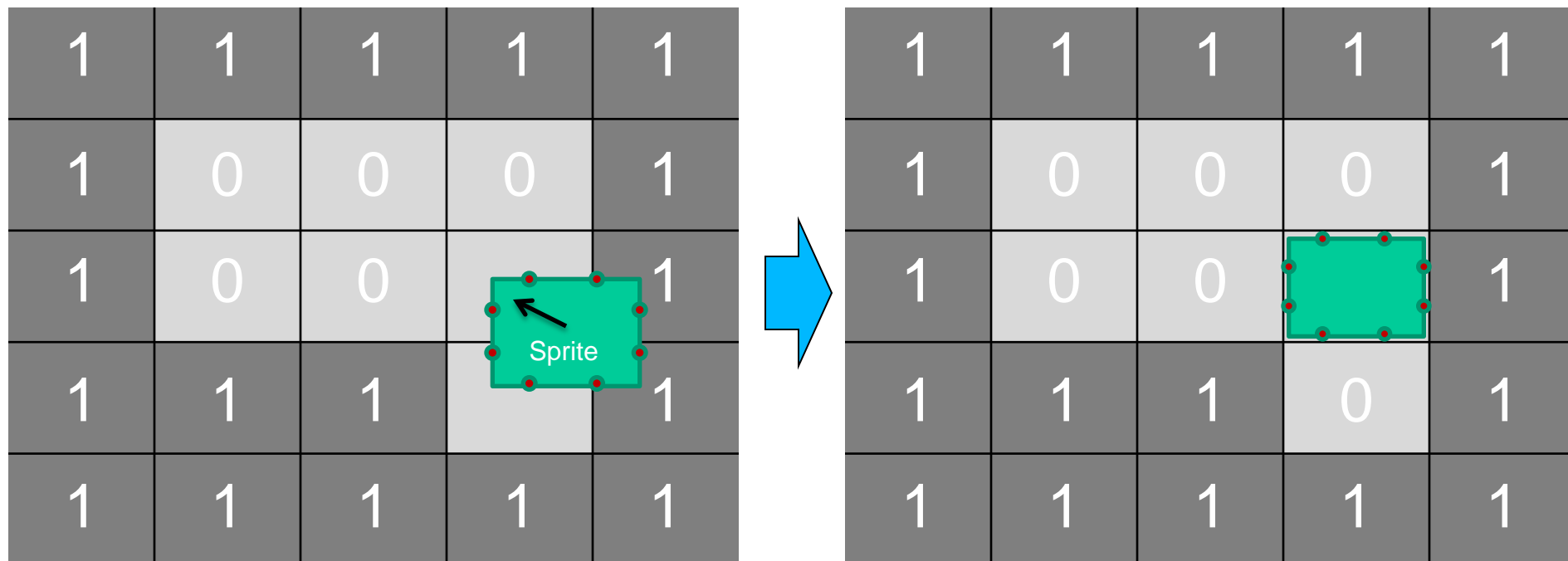


Snapping

- If at least one hot spot is inside a collision area, we should snap the sprite back to the center of the cell that it belongs to.

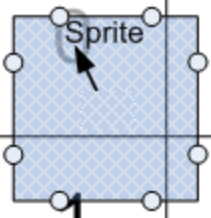
Note: This method assumes that both width and height of an object are equal and have the same size of a tile)

Snapping



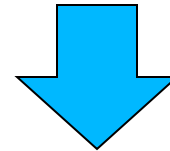
Multiple methods can be used to know which hot spot or side is colliding.

Snapping

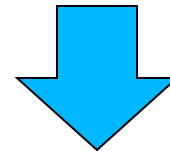
0	0	0	1
0	0	0	1
0	0		1
1	1	1	1

The following is true if:

- Tile's and Object's (0,0) is the top left
- First tile starts at (0,0)
- All tiles and objects have positive X and Y values



$$\begin{aligned}\text{SpriteX} &= \text{XArrayIndex} * \text{TileWidth} \\ \text{SpriteY} &= \text{YArrayIndex} * \text{TileHeight}\end{aligned}$$



$$\begin{aligned}\text{SpriteX} &= \text{XArrayIndex} * 32 = 2 * 32 = 64 \\ \text{SpriteY} &= \text{YArrayIndex} * 32 = 2 * 32 = 64\end{aligned}$$

The End 😊