Platformer games.

Revisiting entities

```
class Entity {
    public:

        void Draw(ShaderProgram &program);

        float x;
        float y;
        float rotation;

        int textureID;


        float width;
        float height;

        float speed;
        float direction_x;
        float direction_y;
};
```

Entities are a useful way for us to think about objects in the game.

```
class Entity {
  public:

  Entity();

  void Update(float elapsed);
  void Render(ShaderProgram &program);
  bool CollidesWith(const Entity &entity);

  SheetSprite sprite;

  float x;
  float y;

  float width;
  float height;

  float velocity_x;
  float velocity_y;

  float acceleration_x;
  float acceleration_y;
};
```

Updating the Entity class.

```
class Entity {
  public:

  Entity();

  void Update(float elapsed);
  void Render(ShaderProgram *program);
  bool CoollidesWith(const Entity &entity);

  SheetSprite sprite;

  Vector3 position;
  Vector3 size;
  Vector3 velocity;
  Vector3 acceleration;

};
```

Updating the Entity class.

Dynamic and static entities.

Dynamic: gravity applied and checking collisions with other entities.

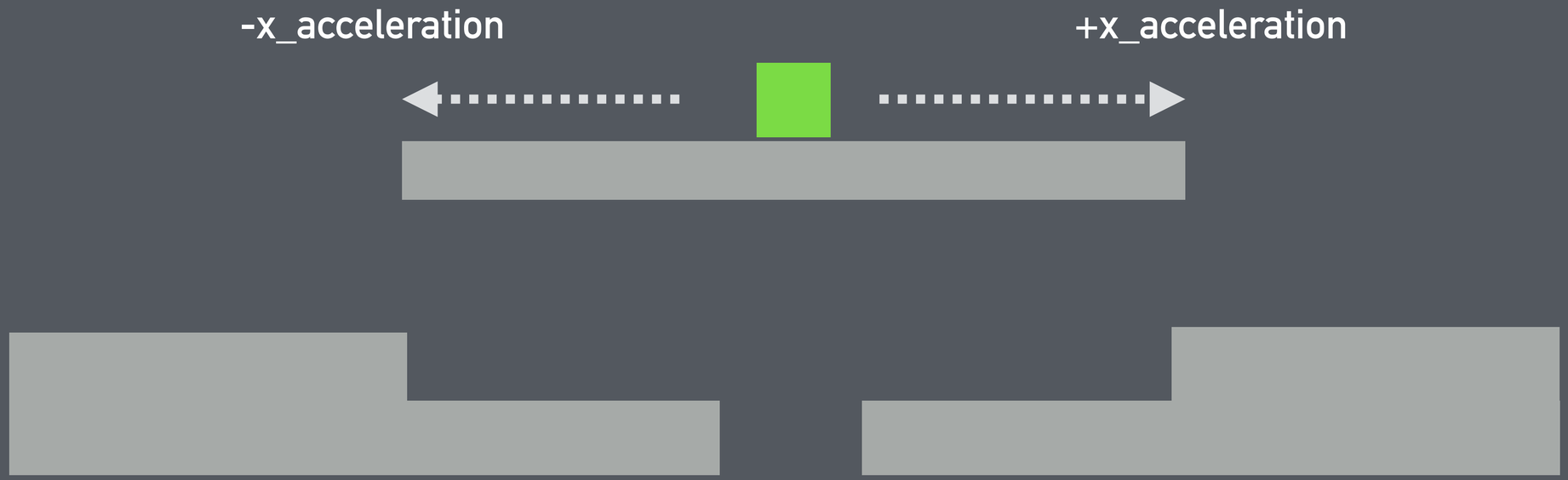
Static: No gravity, no movement, no collision checking!

```
class Entity {
  public:

  Entity();

  void Update(float elapsed);
  void Render(ShaderProgram &program);
  bool CollidesWith(const Entity &entity);

  SheetSprite sprite;

  Vector3 position;
  Vector3 size;
  Vector3 velocity;
  Vector3 acceleration;

  bool isStatic;
};
```

Adding a static flag.

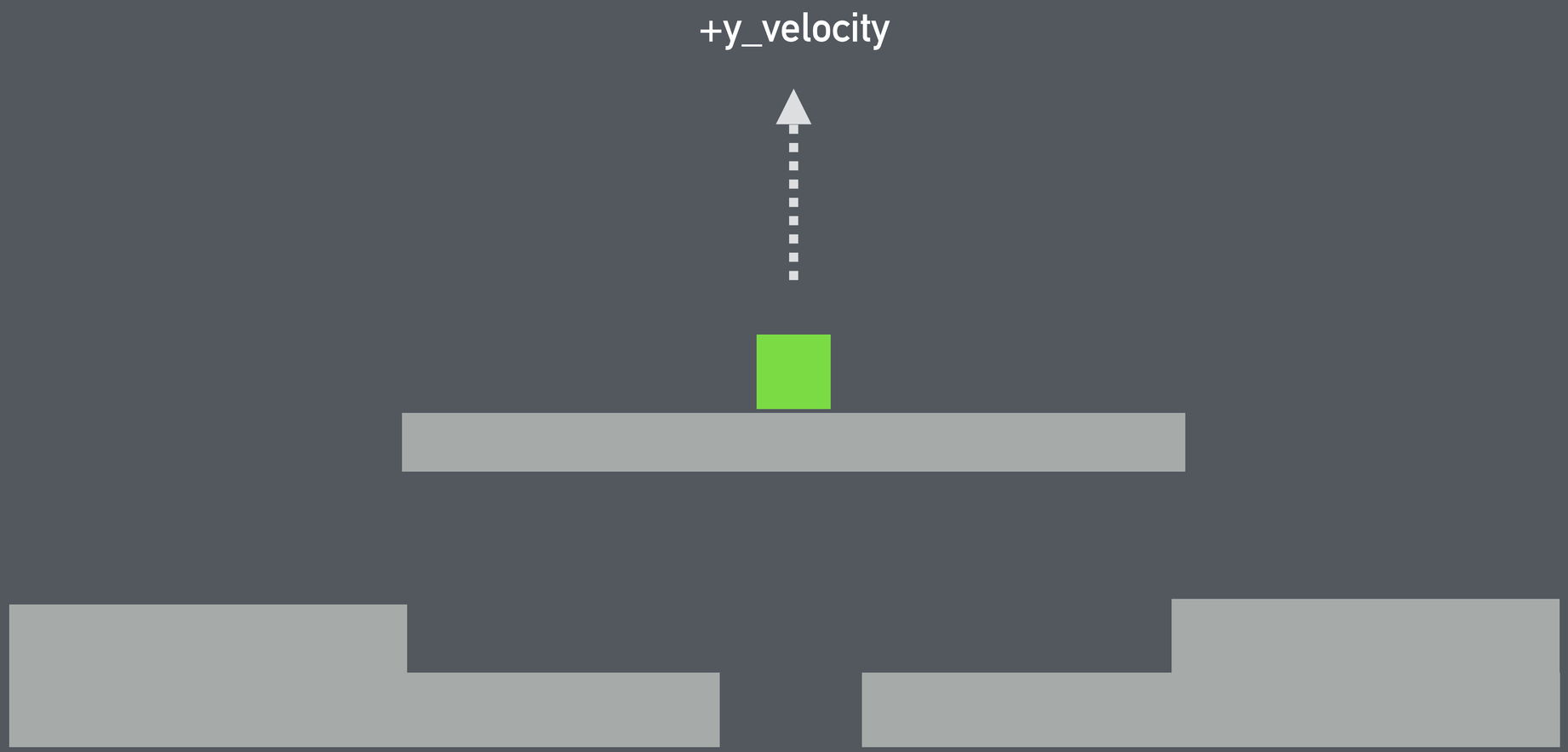
Movement

Set X acceleration to positive or negative to move and to 0 to stop.



Jumping

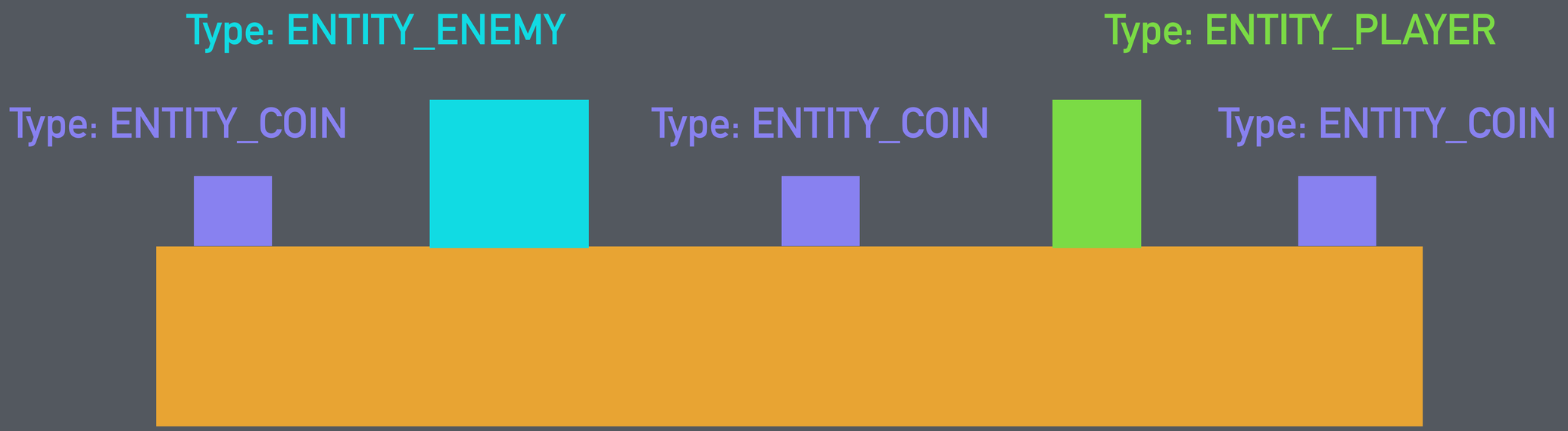
Set Y velocity directly to jump.



Entity types

```
enum EntityType {ENTITY_PLAYER, ENTITY_ENEMY,
ENTITY_COIN};

class Entity {
  public:

  Entity();

  void Update(float elapsed);
  void Render(ShaderProgram *program);
  bool CollidesWith(const Entity &entity);

  SheetSprite sprite;

  Vector3 position;
  Vector3 size;
  Vector3 velocity;
  Vector3 acceleration;
  bool isStatic;

  EntityType entityType;
};
```

Adding an entity type.


Type: ENTITY_ENEMY
Type: ENTITY_PLAYER
Type: ENTITY_COIN
Type: ENTITY_COIN
Type: ENTITY_COIN

Check collision between all dynamic entities and do something based on their type combination.

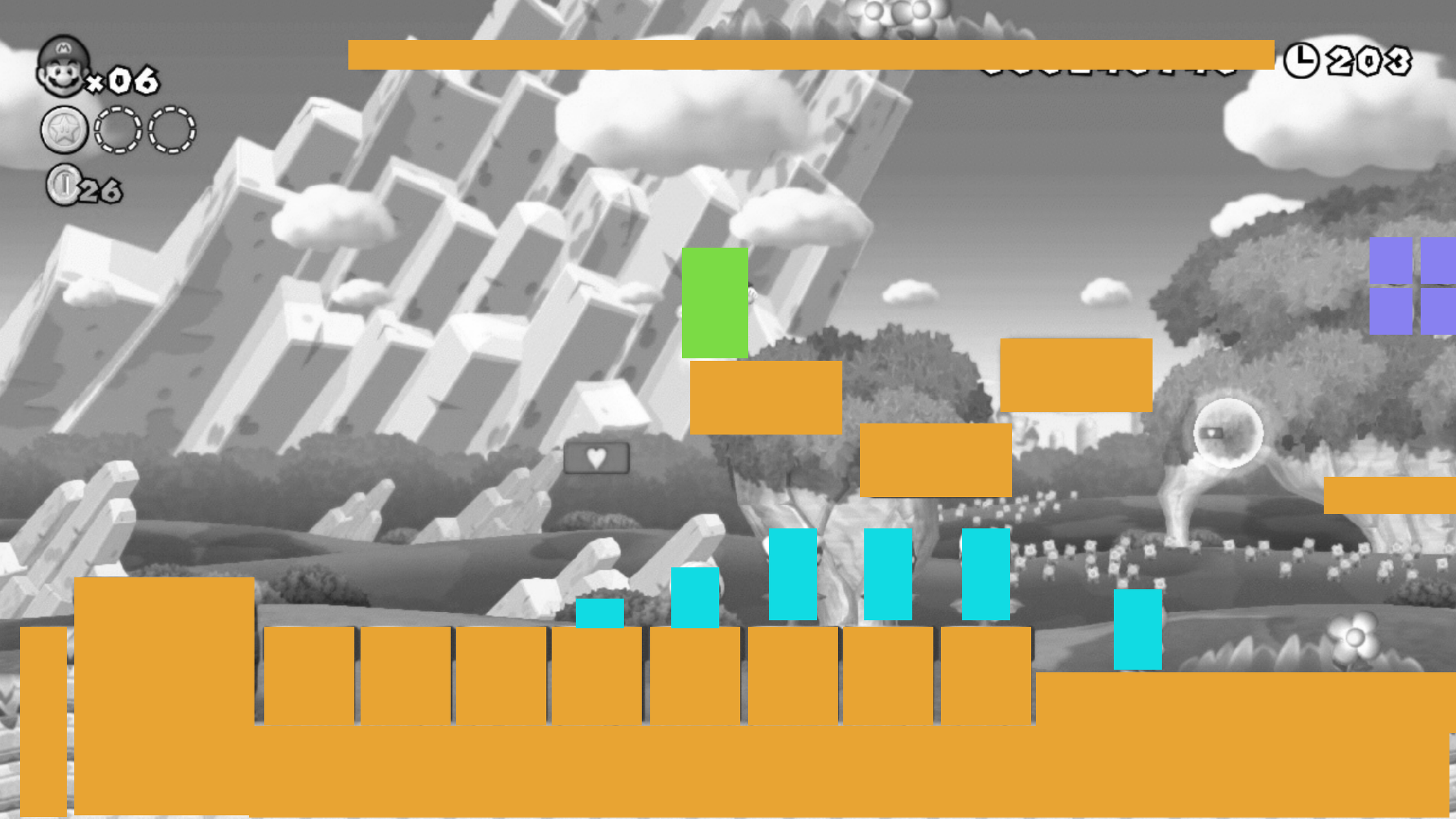
ENTITY_COIN + ENTITY_PLAYER = Make coin entity invisible / remove it and increase player coin count.

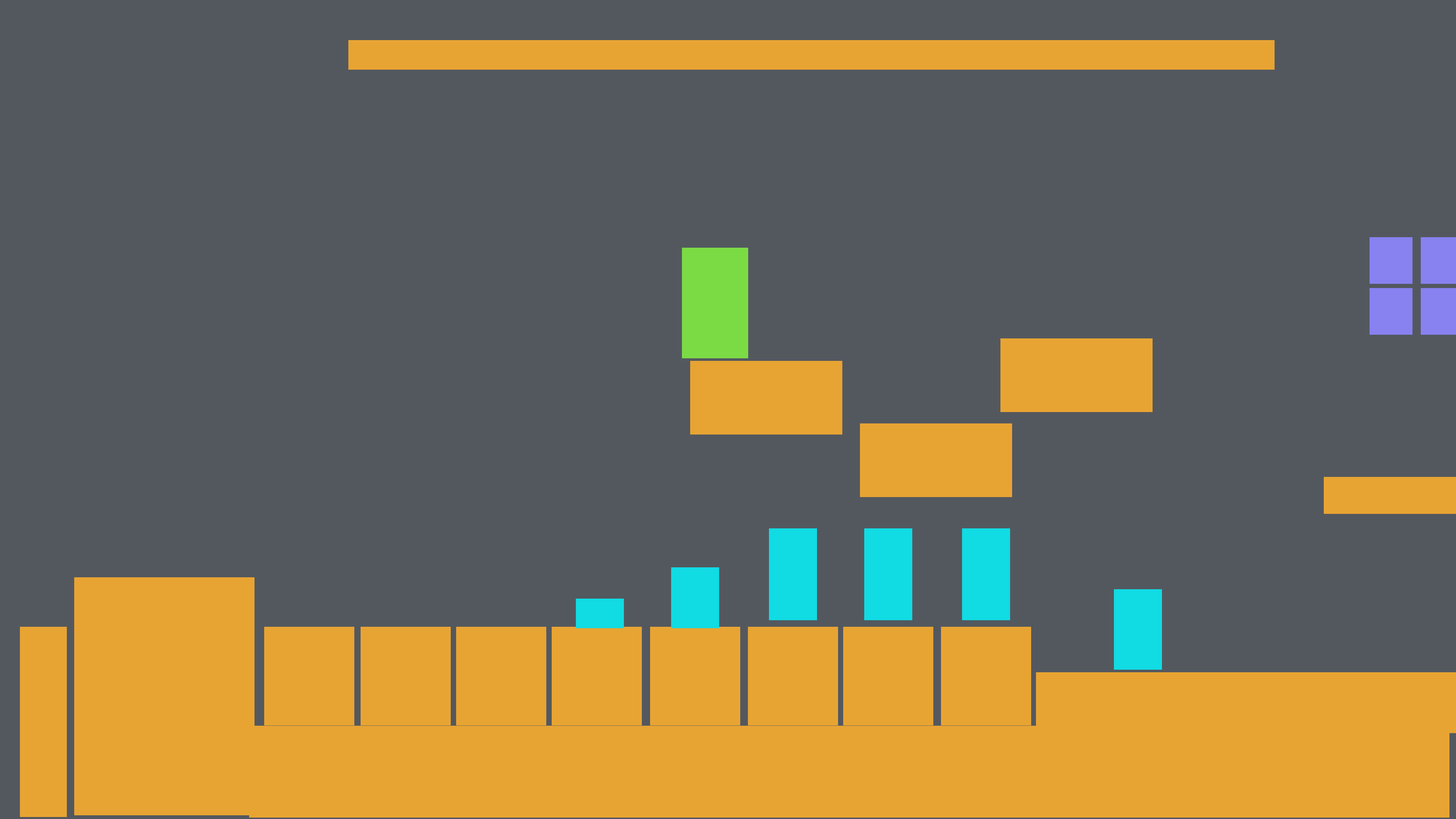
ENTITY_ENEMY + ENTITY_PLAYER = Damage player,

Platformer game breakdown.

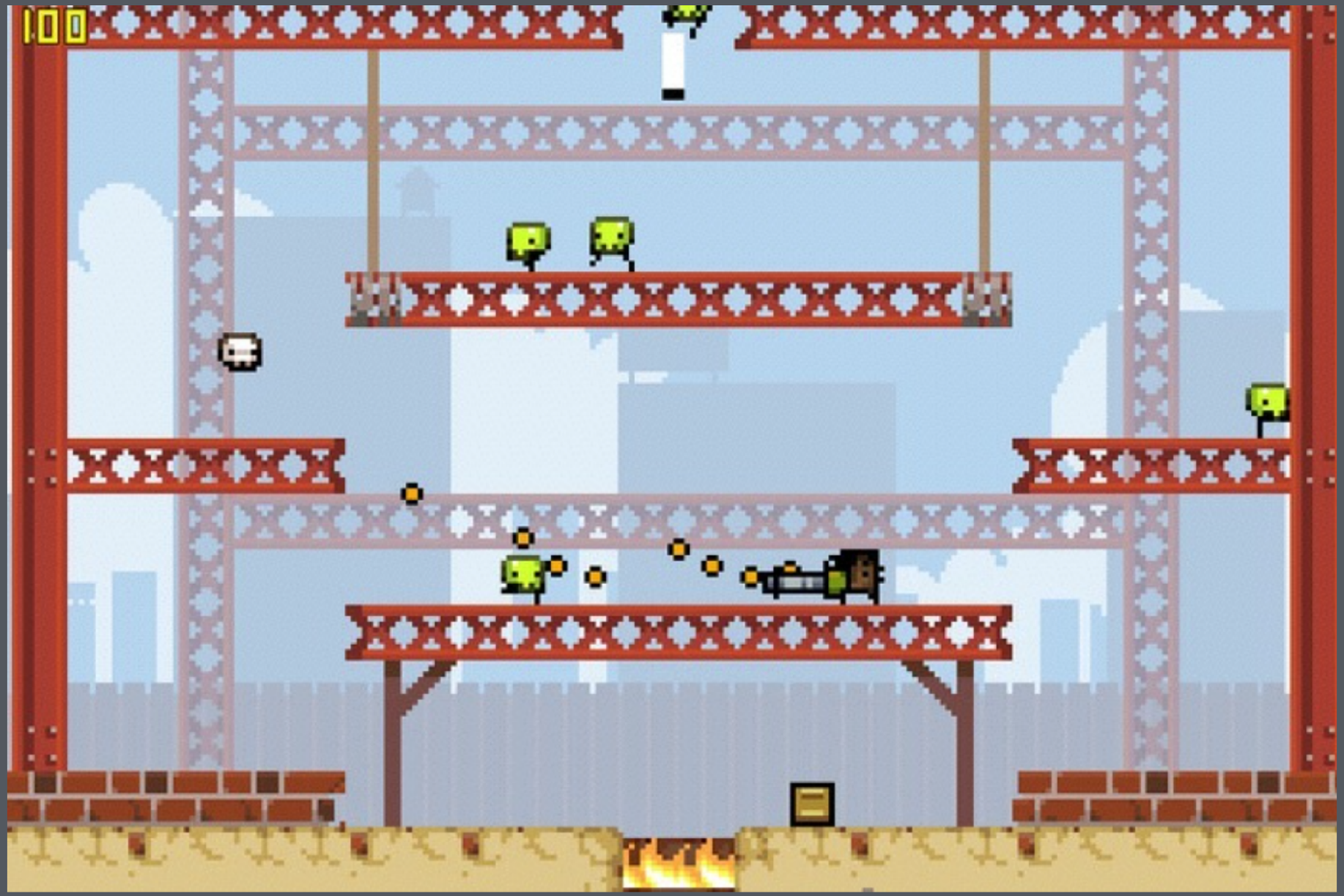

×06
26
000240740
203

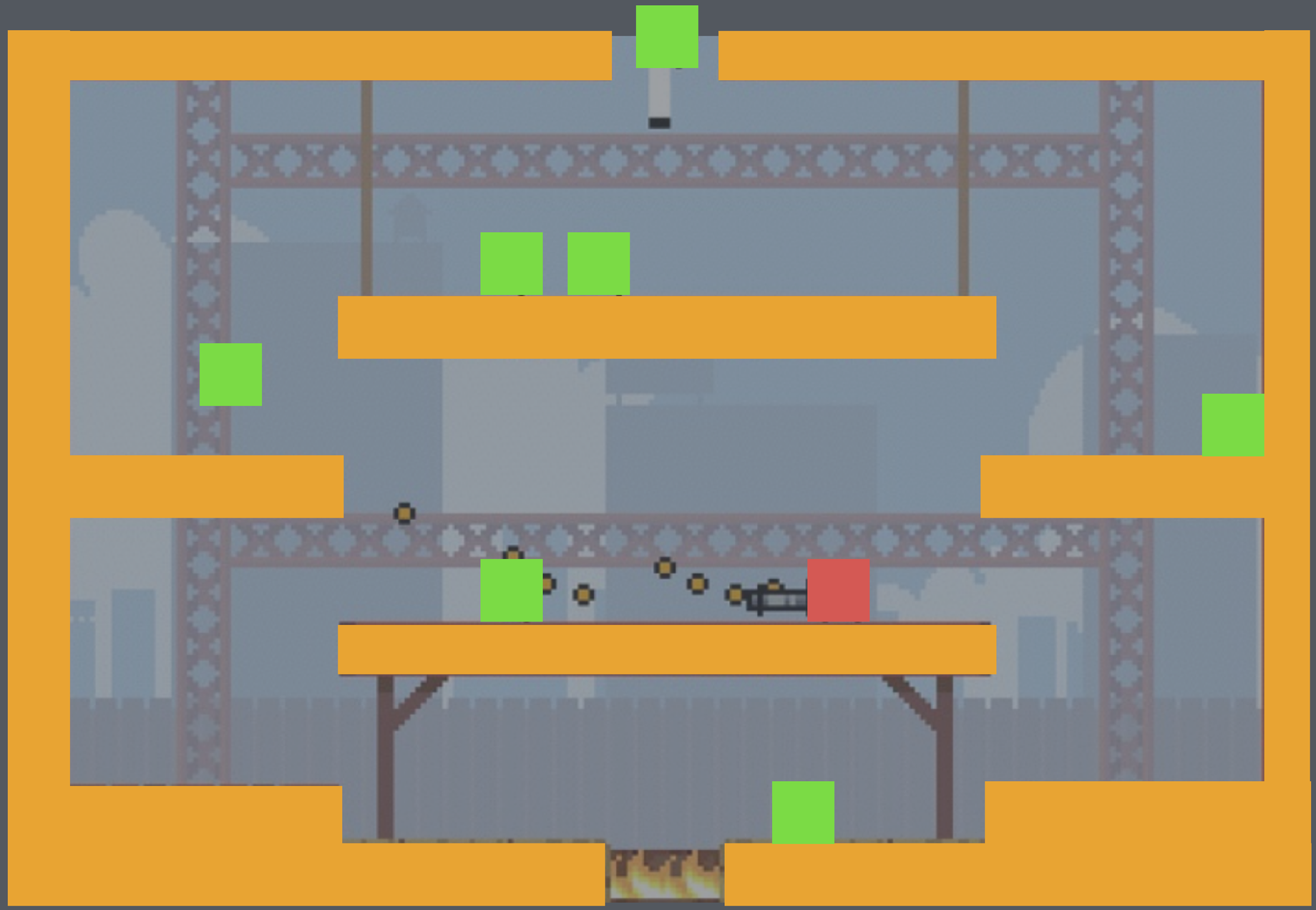

×06
26
203

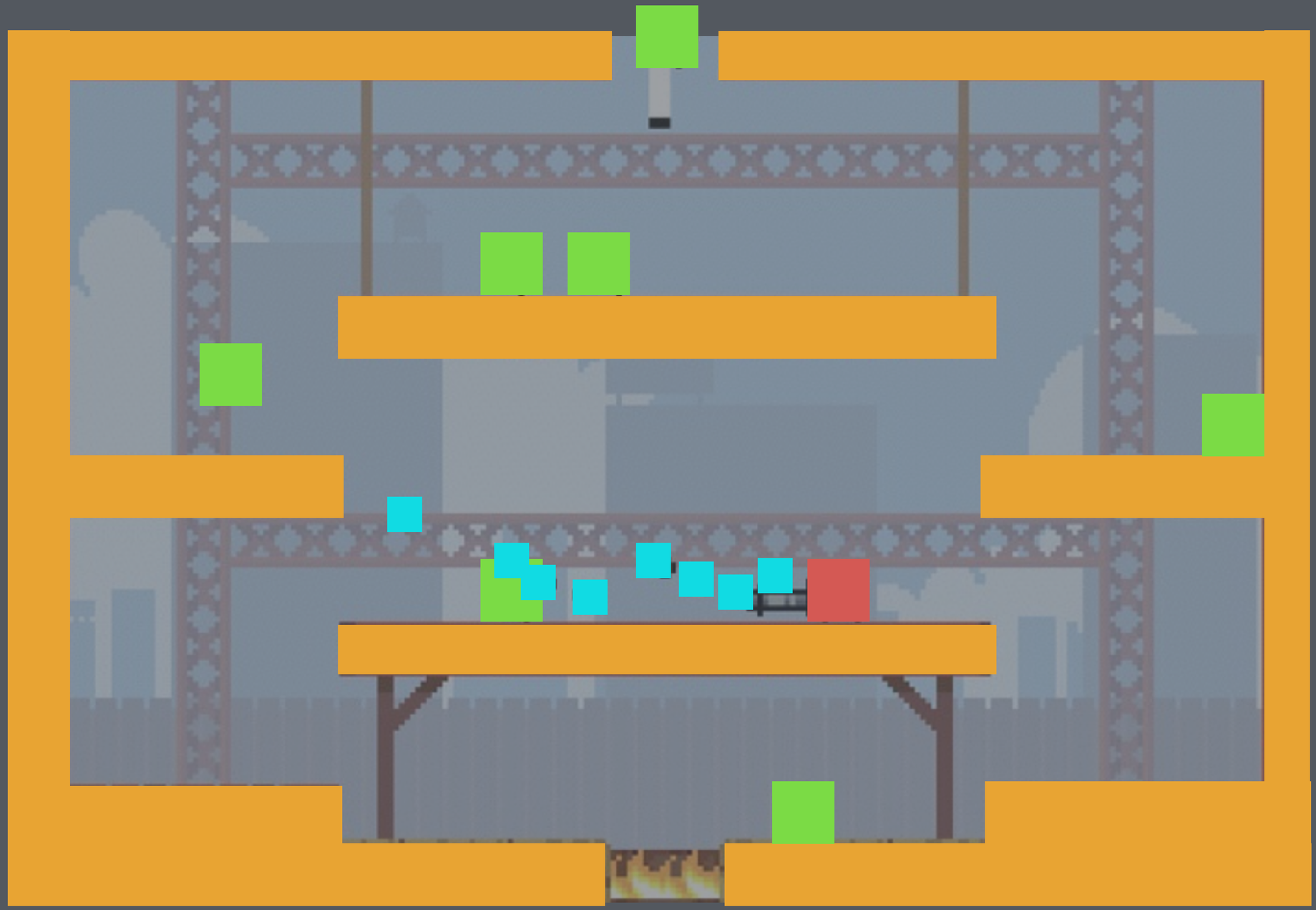

×06
26
203

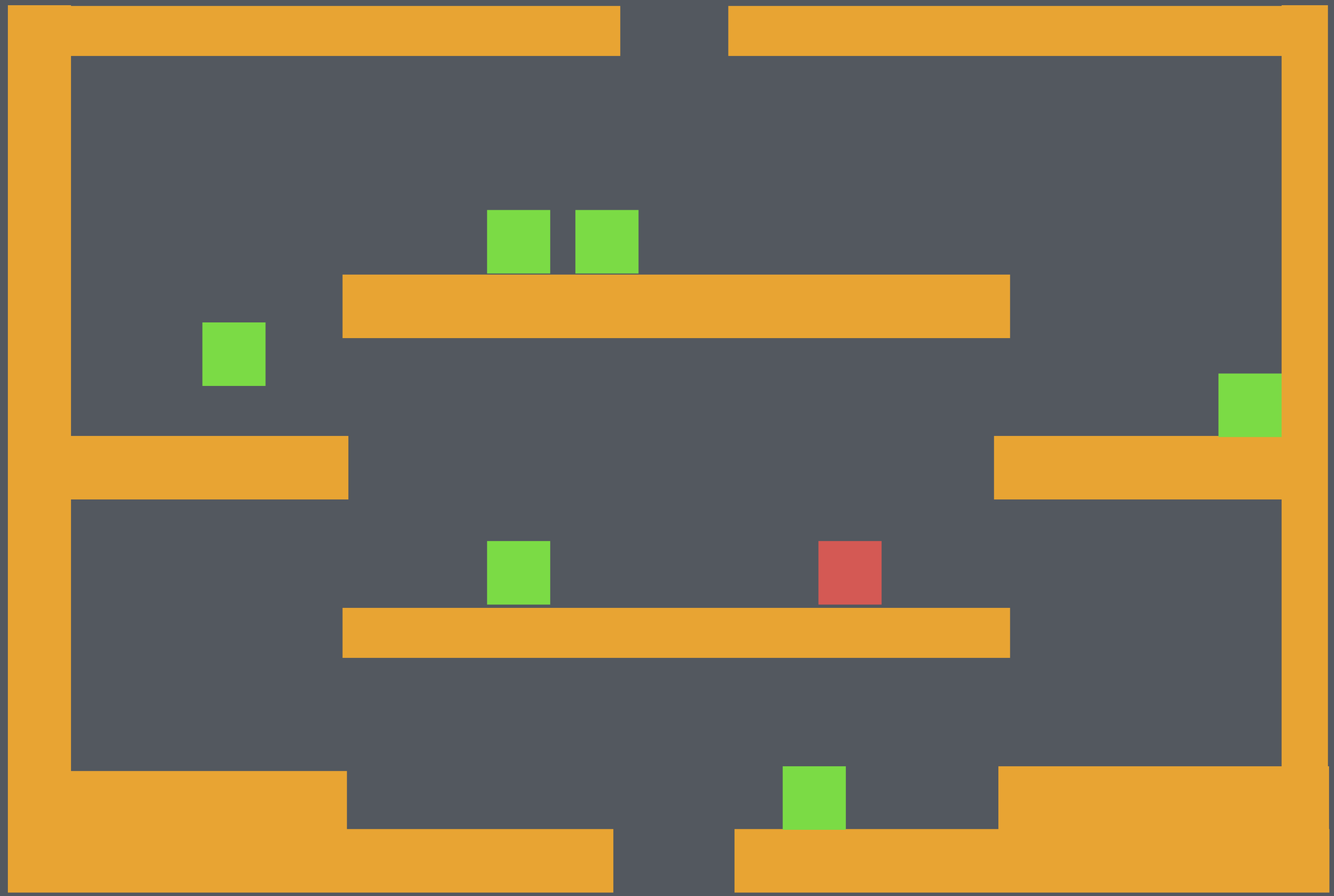


Building a single screen platformer.



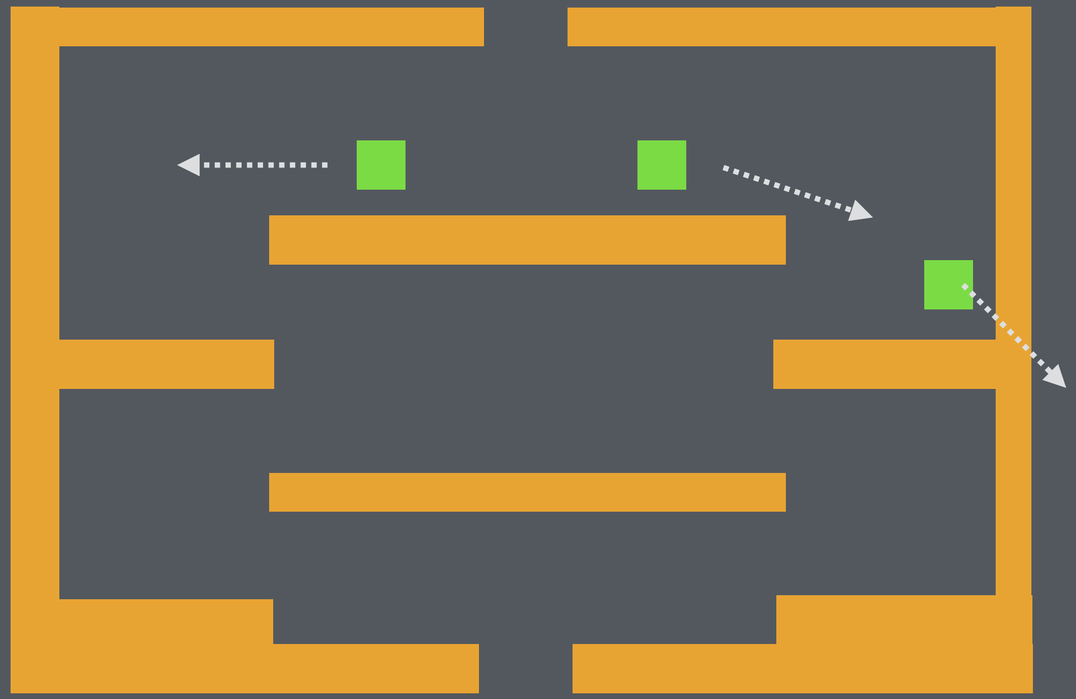


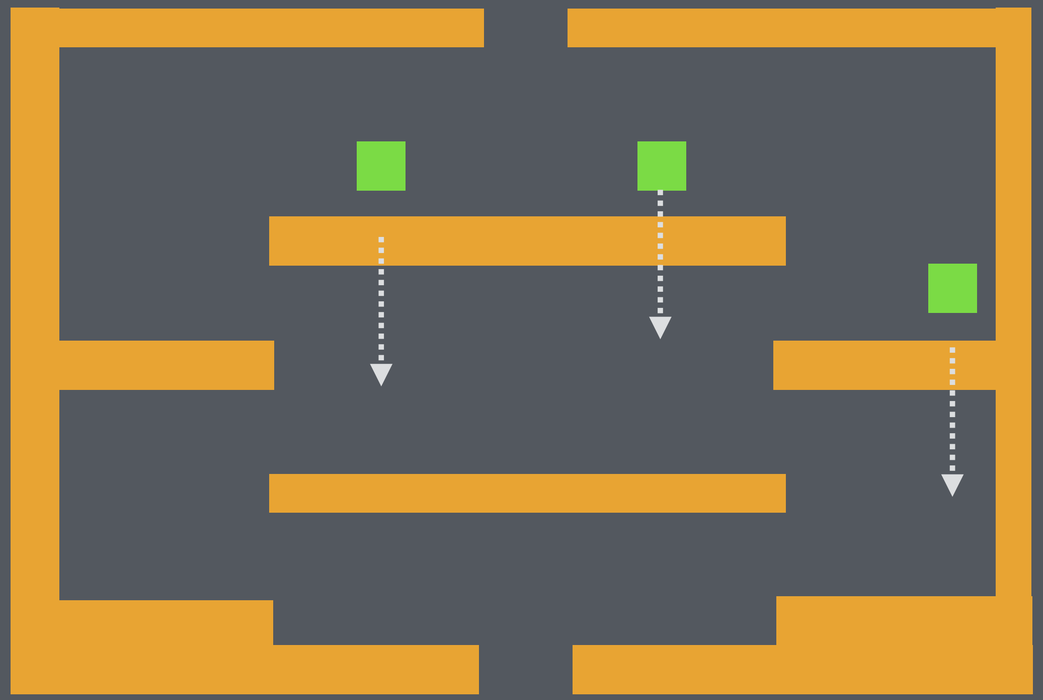




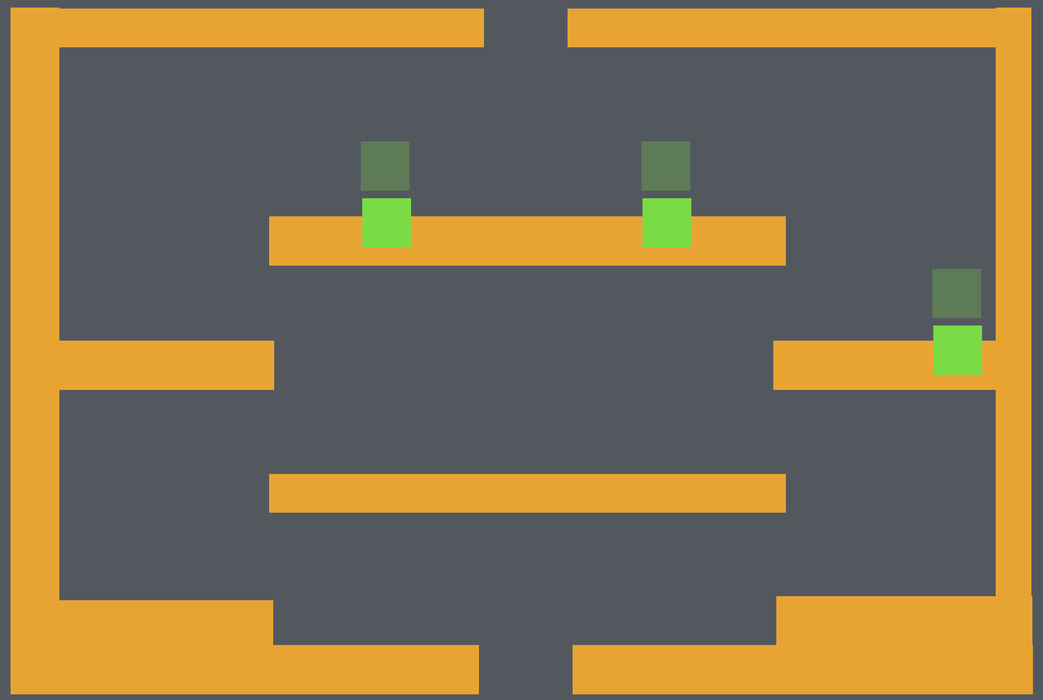
Putting it all together.

1. Apply **acceleration** and **friction** to velocity of **dynamic** entities.

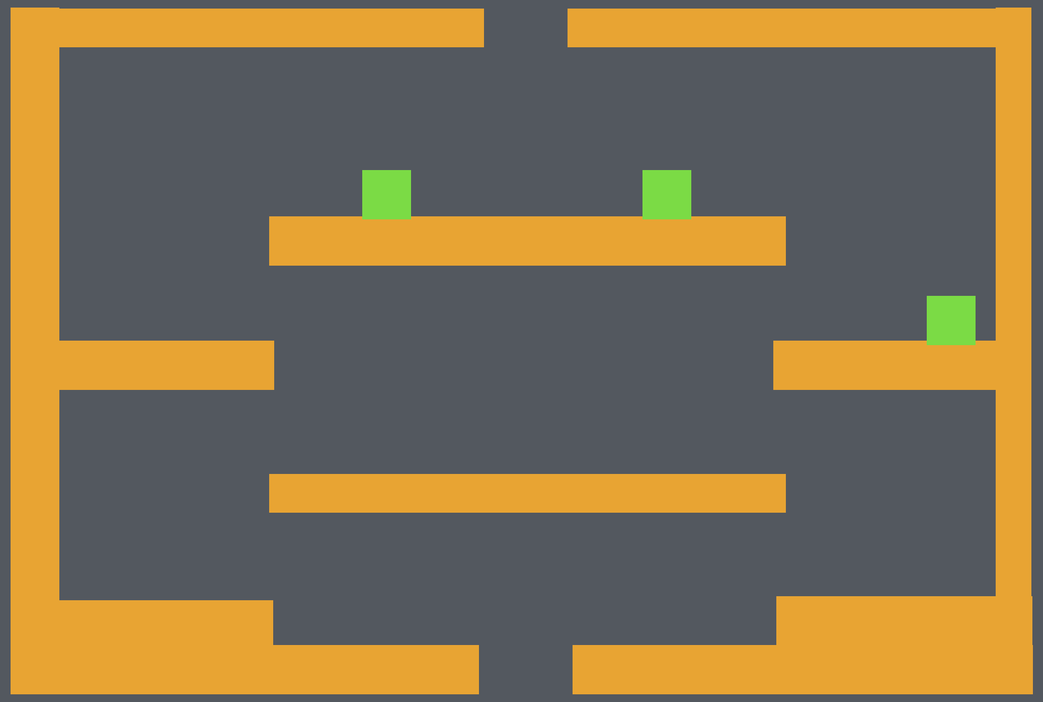




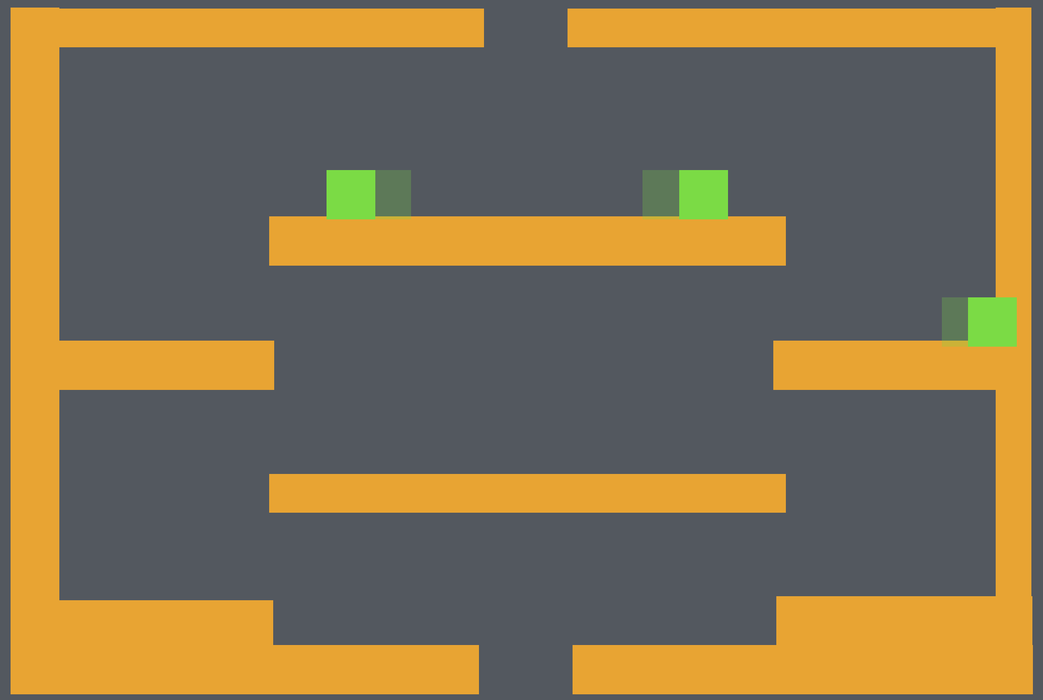
1. Apply **acceleration** and **friction** to velocity of **dynamic** entities.
2. Apply **gravity** to velocity of **dynamic** entities.



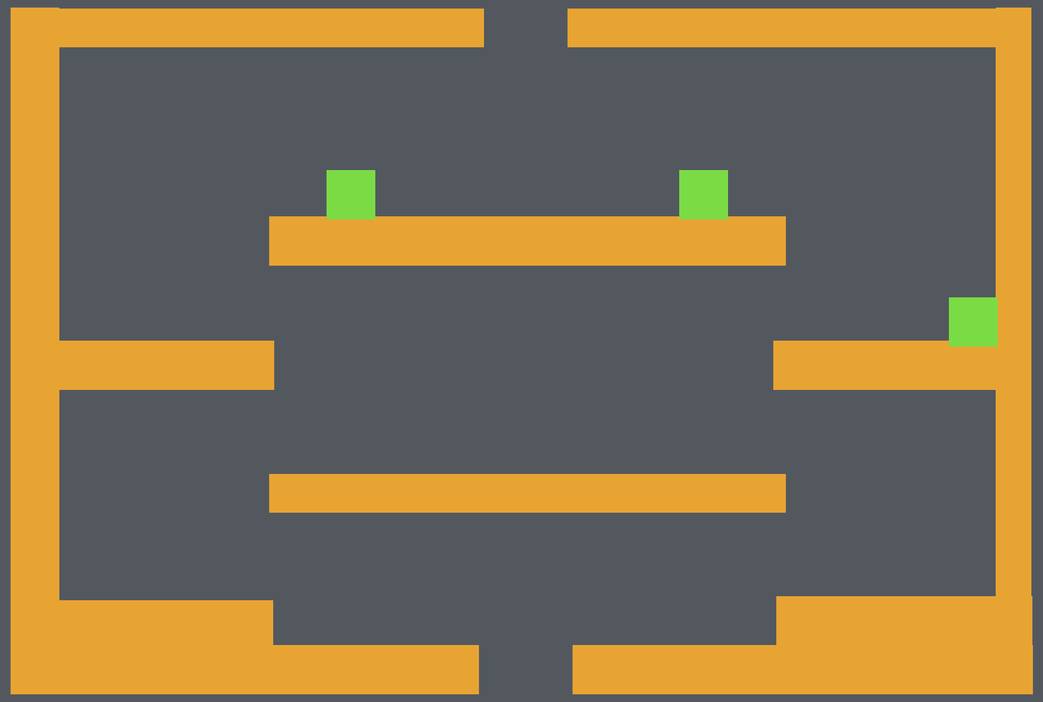
1. Apply **acceleration** and **friction** to velocity of **dynamic** entities.
2. Apply **gravity** to velocity of **dynamic** entities.
3. Apply **ONLY Y-axis velocity** to **dynamic entity** positions.



1. Apply **acceleration** and **friction** to velocity of **dynamic** entities.
2. Apply **gravity** to velocity of **dynamic** entities.
3. Apply **ONLY Y-axis velocity** to **dynamic entity** positions.
4. For each **dynamic entity**, check collisions with **static entities** and adjust **Y-position** based on **penetration**. If **collided,** set **y-velocity** to **0.**



1. Apply **acceleration** and **friction** to velocity of **dynamic** entities.
2. Apply **gravity** to velocity of **dynamic** entities.
3. Apply **ONLY Y-axis velocity** to **dynamic entity** positions.
4. For each **dynamic entity**, check collisions with **static entities** and adjust **Y-position** based on **penetration**. If **collided,** set **y-velocity** to **0.**
5. Apply **ONLY X-axis velocity** to **dynamic entity** positions.



1. Apply **acceleration** and **friction** to velocity of **dynamic** entities.
2. Apply **gravity** to velocity of **dynamic** entities.
3. Apply **ONLY Y-axis velocity** to **dynamic entity** positions.
4. For each **dynamic entity**, check collisions with **static entities** and adjust **Y-position** based on **penetration**. If **collided,** set **y-velocity** to **0.**
5. Apply **ONLY X-axis velocity** to **dynamic entity** positions.
6. For each **dynamic entity**, check collisions with **static entities** and adjust **X-position** based on **penetration**. If **collided,** set **x-velocity** to **0.**

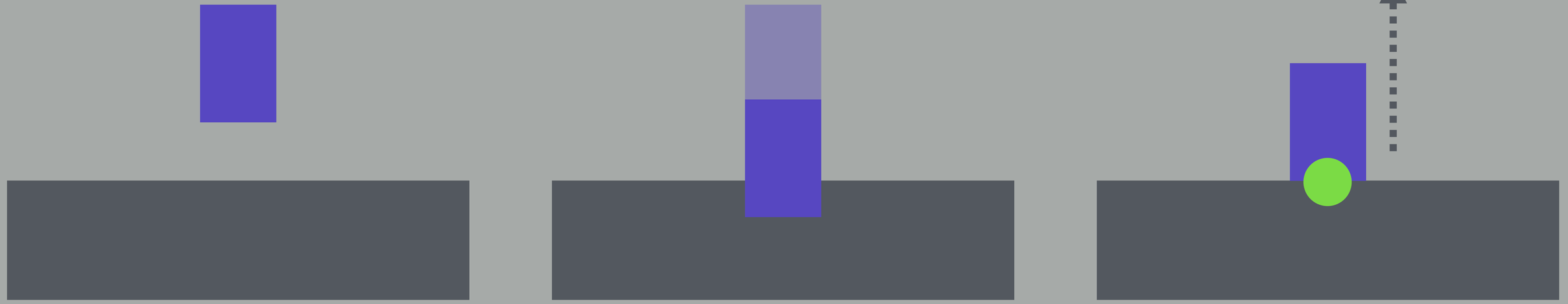
Contact flags

4 boolean flags, one for each side.

```
bool collidedTop;
bool collidedBottom;
bool collidedLeft;
bool collidedRight;
```

```
collidedTop = false;
collidedBottom = false;
collidedLeft = false;
collidedRight = false;
```

Set to false on every frame.



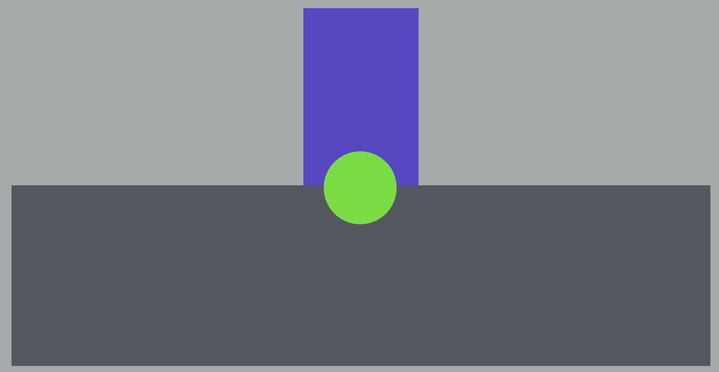
```
collidedBottom = true;
```

Set to true based on collision direction.

Using contact flags

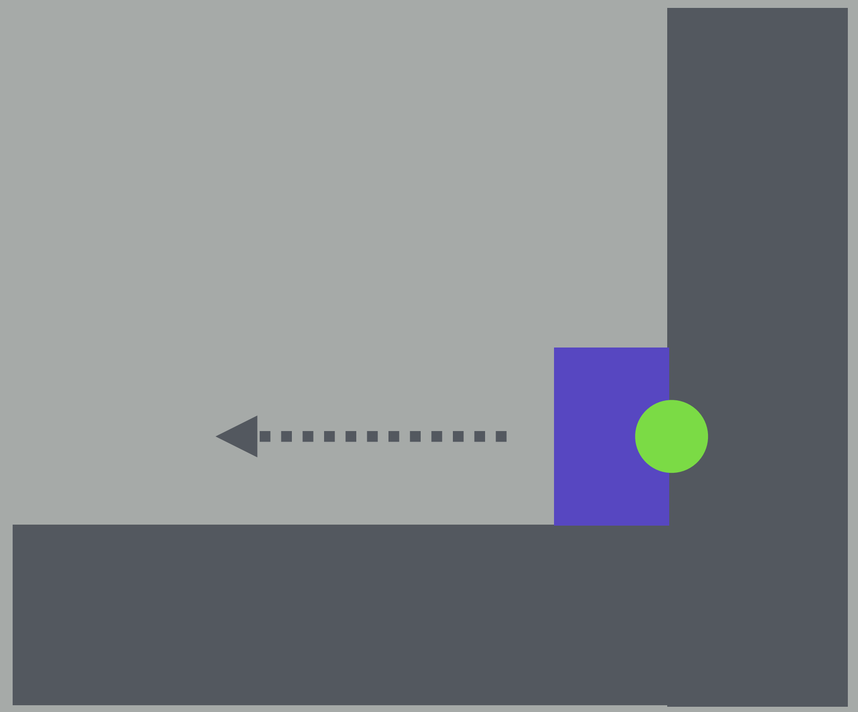
Detecting if you are standing on the ground.

(Jump only when bottom contact flag is true)



Simple NPC entities that turn around when they hit a wall.

(Constantly apply **x acceleration**, when **left or right contact** flag is **true,** reverse **the x acceleration**)



```
enum EntityType {ENTITY_PLAYER, ENTITY_ENEMY,
ENTITY_COIN};

class Entity {
  public:

  Entity();

  void Update(float elapsed);
  void Render(ShaderProgram &program);
  bool CollidesWith(const Entity &entity);

  SheetSprite sprite;

  Vector3 position;
  Vector3 size;
  Vector3 velocity;
  Vector3 acceleration;

  bool isStatic;
  EntityType entityType;

  bool collidedTop;
  bool collidedBottom;
  bool collidedLeft;
  bool collidedRight;
};
```

Final entity properties.