Basic Game Physics

Gravity
Jumping
Movement
(somewhat automagically)

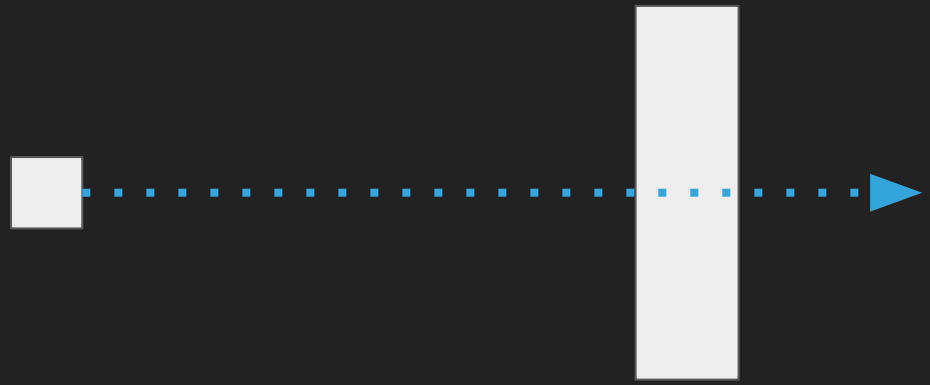
Topics:
Fixed Timestep
Velocity
Acceleration (Gravity)

Currently, our timestep is as fast as our computer can go as well as variable.

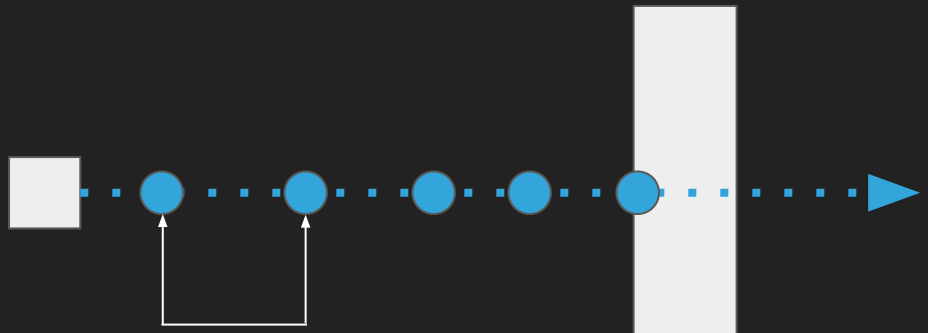
```
void Update() {
    float ticks = (float)SDL_GetTicks() / 1000.0f;
    float deltaTime = ticks - lastTicks;
    lastTicks = ticks;

    // Add (direction * units per second * elapsed time)
    player_position += player_movement * player_speed * deltaTime;
}
```

Variable Timestep

Variable Timestep

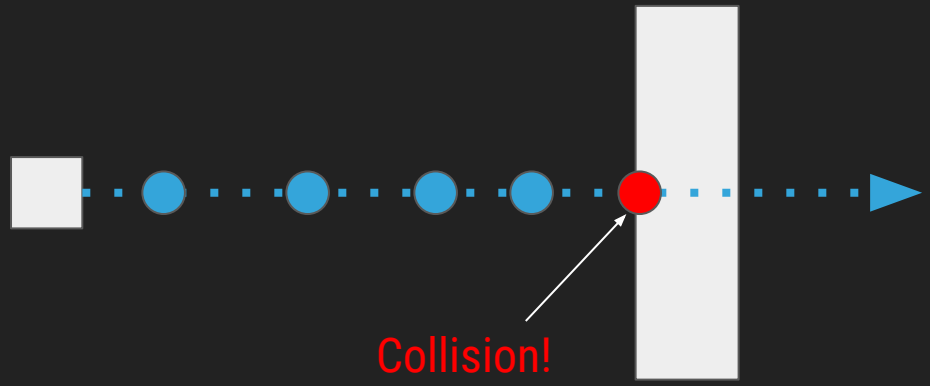


Variable Timestep



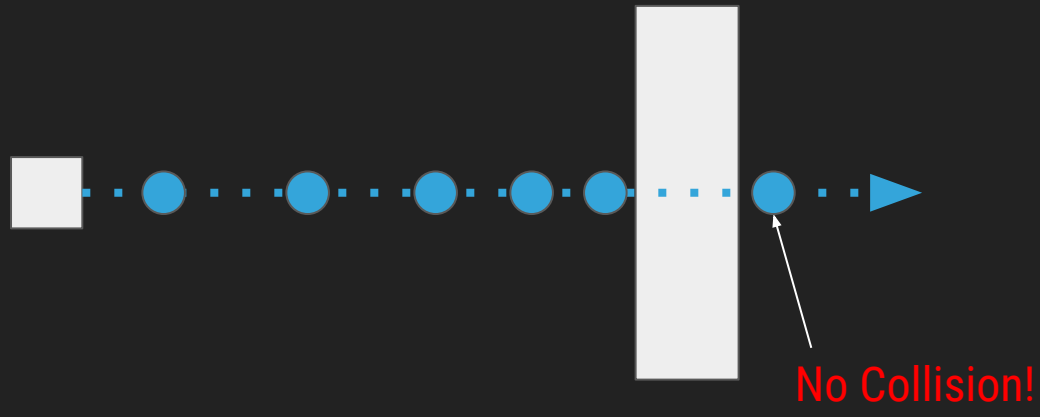
Elapsed time since last frame

Variable Timestep



(everything worked out OK here)

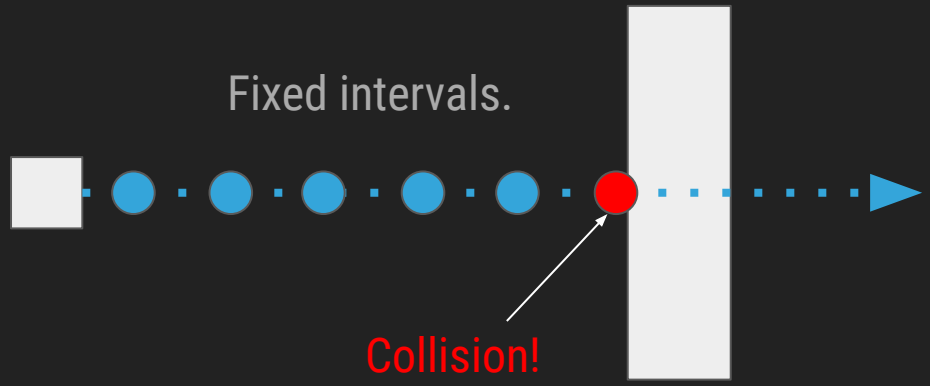
Variable Timestep



Due to our variable timestep,
we “skipped over” the object.

To keep physics behaviors the same, we want to use a fixed timestep.

Fixed Timestep



(everything worked out OK here)

```
#define FIXED_TIMESTEP 0.0166666f
float lastTicks = 0;
float accumulator = 0.0f;

void Update() {
    float ticks = (float)SDL_GetTicks() / 1000.0f;
    float deltaTime = ticks - lastTicks;
    lastTicks = ticks;

    deltaTime += accumulator;
    if (deltaTime < FIXED_TIMESTEP) {
        accumulator = deltaTime;
        return;
    }

    while (deltaTime >= FIXED_TIMESTEP) {
        // Update. Notice it's FIXED_TIMESTEP. Not deltaTime
        state.player->Update(FIXED_TIMESTEP);

        deltaTime -= FIXED_TIMESTEP;
    }

    accumulator = deltaTime;
}
```

Gravity

(Acceleration due to Gravity)

$9.81\ m/s^2$

Gravity

(Acceleration due to Gravity)

```
player.acceleration = glm::vec3(0, -9.81f, 0);
```

Acceleration

Rate of change of velocity.

```
velocity.x += acceleration.x * elapsed;
velocity.y += acceleration.y * elapsed;

// You can also do this
velocity += acceleration * elapsed;
```

Velocity

Change of position over time.

```
position.x += velocity.x * elapsed;
position.y += velocity.y * elapsed;

// You can also do this
position += velocity * elapsed;
```

Putting it all together:

```
player.acceleration = glm::vec3(0, -9.81f, 0);

void Update(float deltaTime) {  // player’s update
    velocity += acceleration * deltaTime;
    position += velocity * deltaTime;
}
```

Notice if **acceleration** never changes,
velocity will keep accumulating.

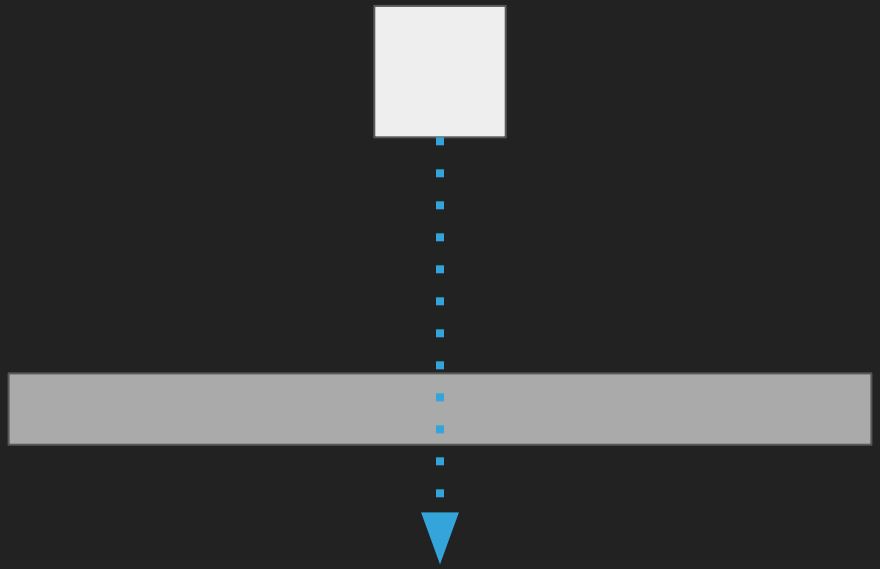
Let’s Code!

Example: FixedTimeStep

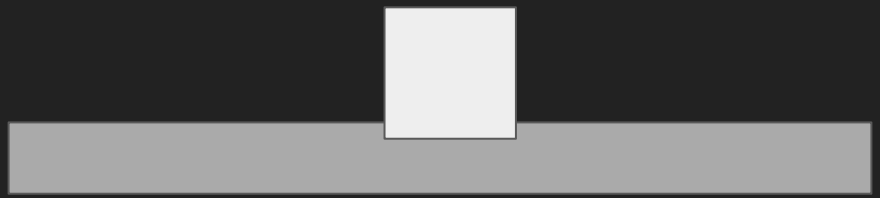
Add Physics
Add Platforms
Add Collision Detection

We Got Stuck!

Check for Overlap

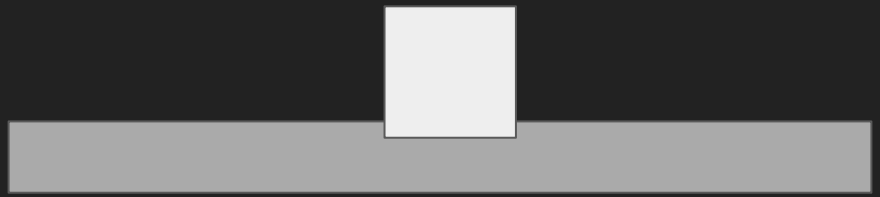


Check for Overlap



Fix Before Rendering

Check for Overlap



```
float ydist = fabs(position.y - other->position.y);
float penetrationY = fabs(ydist - height / 2 - other->height / 2);
```

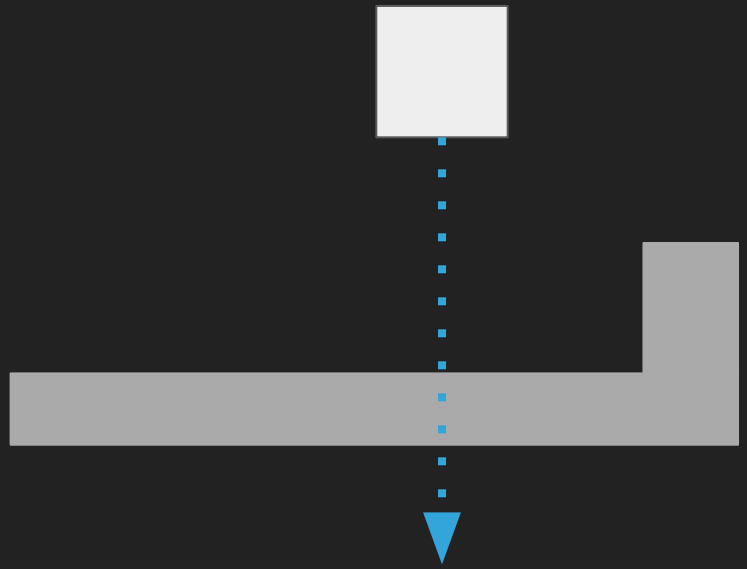
```
void Entity::Update(float deltaTime, Entity *platforms, int platformCount)
{
    velocity += acceleration * deltaTime;
    position += velocity * deltaTime;

    for (int i = 0; i < platformCount; i++)
    {
        Entity *platform = &platforms[i];

        if (CheckCollision(platform))
        {
            float ydist = fabs(position.y - platform->position.y);
            float penetrationY = fabs(ydist - (height / 2.0f) - (platform->height / 2.0f));
            if (velocity.y > 0) {
                position.y -= penetrationY;
                velocity.y = 0;
            } else if (velocity.y < 0) {
                position.y += penetrationY;
                velocity.y = 0;
            }
        }
    }
}
```

Let’s Code!

Update Collision Detection!
Add Jumping!
Test Moving!

We need to change our collision detection algorithm.

Use Y velocity first...

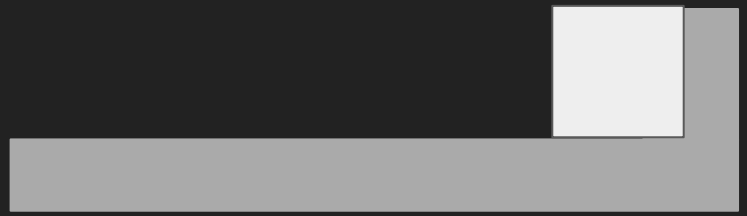


Check for collisions...

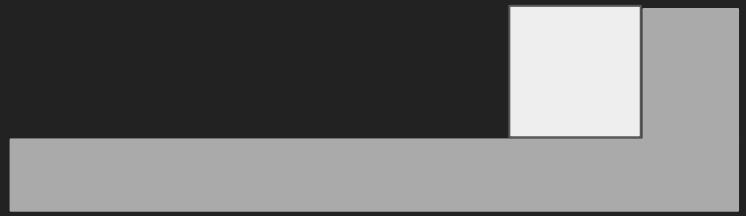
Adjust based on penetration...

Use X velocity next...

Check for collisions...



Adjust based on penetration.



Update Code

```
void Entity::Update(float deltaTime, Entity *platforms, int platformCount)
{
    velocity += acceleration * deltaTime;

    position.y += velocity.y * deltaTime;      // Move on Y
    CheckCollisionsY(platforms, platformCount);// Fix if needed

    position.x += velocity.x * deltaTime;      // Move on X
    CheckCollisionsX(platforms, platformCount);// Fix if needed
}
```

```
void Entity::CheckCollisionsY(Entity *objects, int objectCount)
{
    for (int i = 0; i < objectCount; i++)
    {
        Entity *object = &objects[i];

        if (CheckCollision(object))
        {
            float ydist = fabs(position.y - object->position.y);
            float penetrationY = fabs(ydist - (height / 2.0f) - (object->height / 2.0f));
            if (velocity.y > 0) {
                position.y -= penetrationY;
                velocity.y = 0;
            }
            else if (velocity.y < 0) {
                position.y += penetrationY;
                velocity.y = 0;
            }
        }
    }
}
```

```
void Entity::CheckCollisionsX(Entity *objects, int objectCount)
{
    for (int i = 0; i < objectCount; i++)
    {
        Entity *object = &objects[i];

        if (CheckCollision(object))
        {
            float xdist = fabs(position.x - object->position.x);
            float penetrationX = fabs(xdist - (width / 2.0f) - (object->width / 2.0f));
            if (velocity.x > 0) {
                position.x -= penetrationX;
                velocity.x = 0;
            }
            else if (velocity.x < 0) {
                position.x += penetrationX;
                velocity.x = 0;
            }
        }
    }
}
```

Let’s Code!

CheckCollisionY
CheckCollisionX
Update

Entity Type

Entity Type

```
enum EntityType { PLAYER, PLATFORM, COIN, ENEMY };

class Entity {
public:

    EntityType entityType;

    glm::vec3 position;
    glm::vec3 velocity;
    glm::vec3 acceleration;
```

Entity Type and Update

```
void Entity::Update(float deltaTime, Entity *objects, int objectCount)
{
    if (entityType == WALL) {
        return;
    }
    else if (entityType == COIN) {
        // spin
    }
    else if (entityType == ENEMY) {
        // Move left to right
    }
    else if (entityType == PLAYER) {
        // Do all the things
    }
}
```

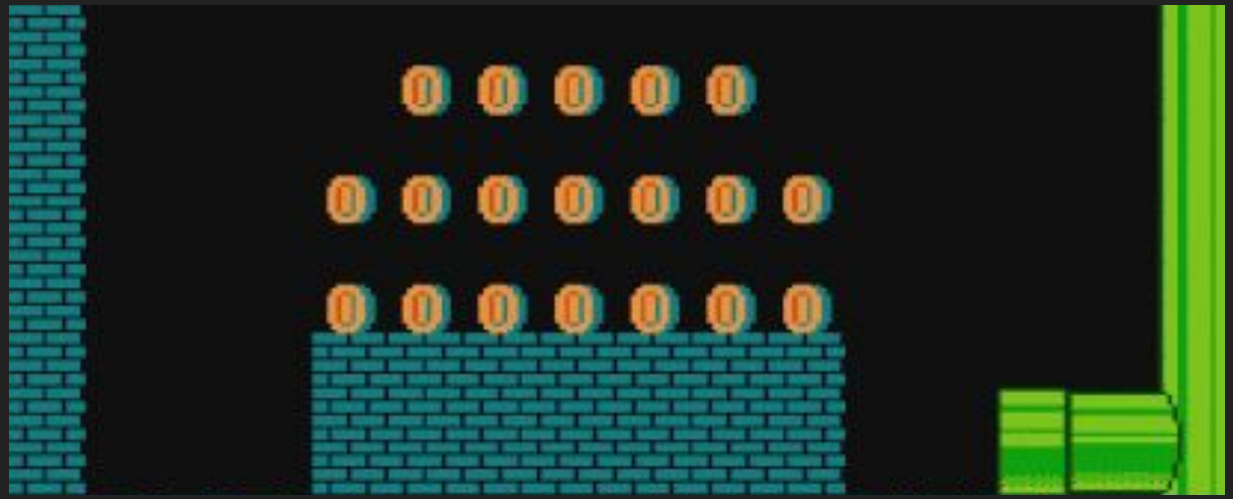
Entity Type and Collision

```
bool Entity::CheckCollision(Entity *other)
{
    float xdist = fabs(position.x - other->position.x) - ((width + other->width) / 2.0f);
    float ydist = fabs(position.y - other->position.y) - ((height + other->height) / 2.0f);

    if (xdist < 0 && ydist < 0)
    {
        lastCollision = other->entityType;
        return true;
    }

    return false;
}
```

Entity Type and Collision

```
// Somewhere in your code

if (player->lastCollision == COIN) {
    // get points
}

else if (player->lastCollision == ENEMY) {
    // take damage
}
```

isActive



isActive

```
class Entity {
public:

    EntityType entityType;
    bool isActive;

    glm::vec3 position;
    glm::vec3 velocity;
    glm::vec3 acceleration;
```

isActive

(collected coins, squashed enemies, objects in object pool)

Update

Exit right away.

CheckCollision

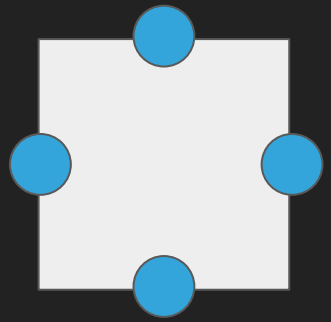
Always false if either object is false!

```
if (isActive == false || other.isActive == false) return false;
```

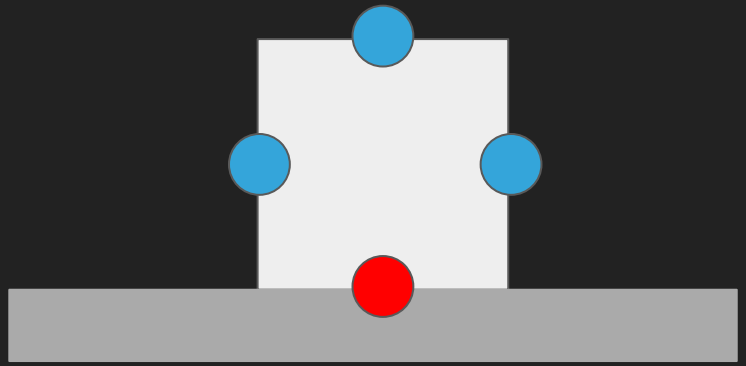
Render

No rendering. Exit right away.

Collision Flags

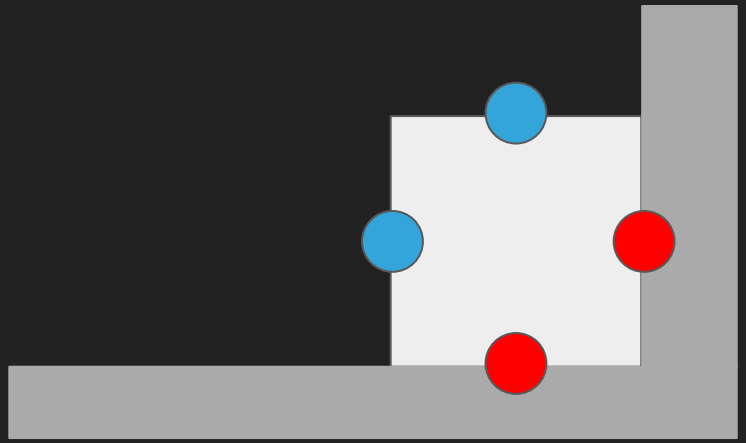


Collision Flags



Player should only be able to
jump when touching the ground.

Collision Flags



Enemies change direction
after hitting a wall.

Collision Flags

```
class Entity {
public:

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

Let’s Code!

isActive
Collision Flags