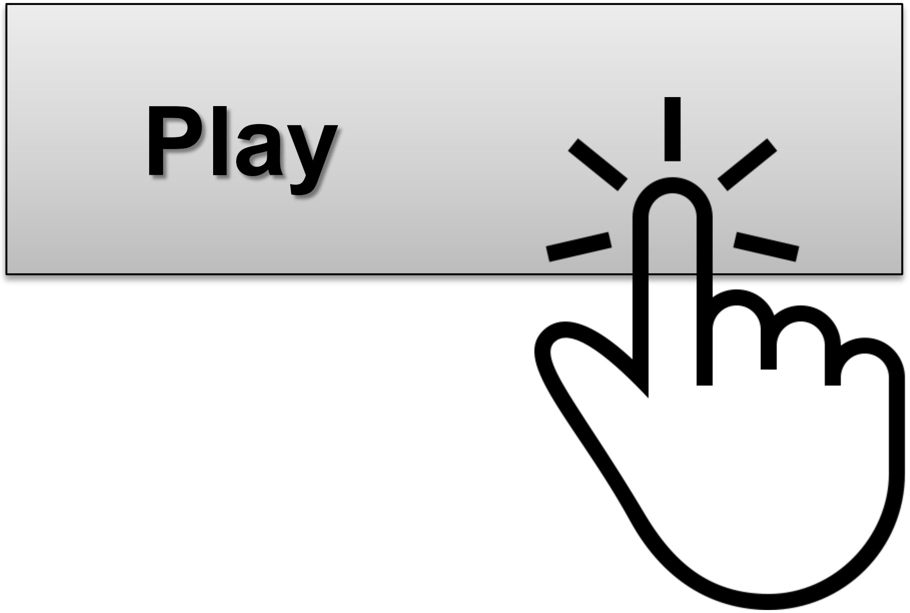
CPSC 427
Video Game Programming

Game Programming Basics: Event Driven Programming & Entity Component System (ECS)



This year’s game theme

~~A) Non violent games, for ‘kids’~~

B) Randomness and surprise!

~~C) Time counts, 10 seconds!~~

Register your Team!

Even if incomplete, please register

-> Canvas -> People -> Groups -> Team

- ***11 people still without a team***
- ***6 teams with 5/6 members***

-> need to form one more team and add 5-6 members

CPSC 427
Video Game Programming

Entity Component System (ECS)

Summary and extensions

ECS is used in Minecraft and many other commercial games

Problem: associating entities and components

	Position	Velocity	Jumps	Player	Squishable
Mario	□	□	□	□	
Goomba1	□	□			□
Luigi	□	□	□		
Goomba2	□	□			□

Object-oriented-programming (OOP)?

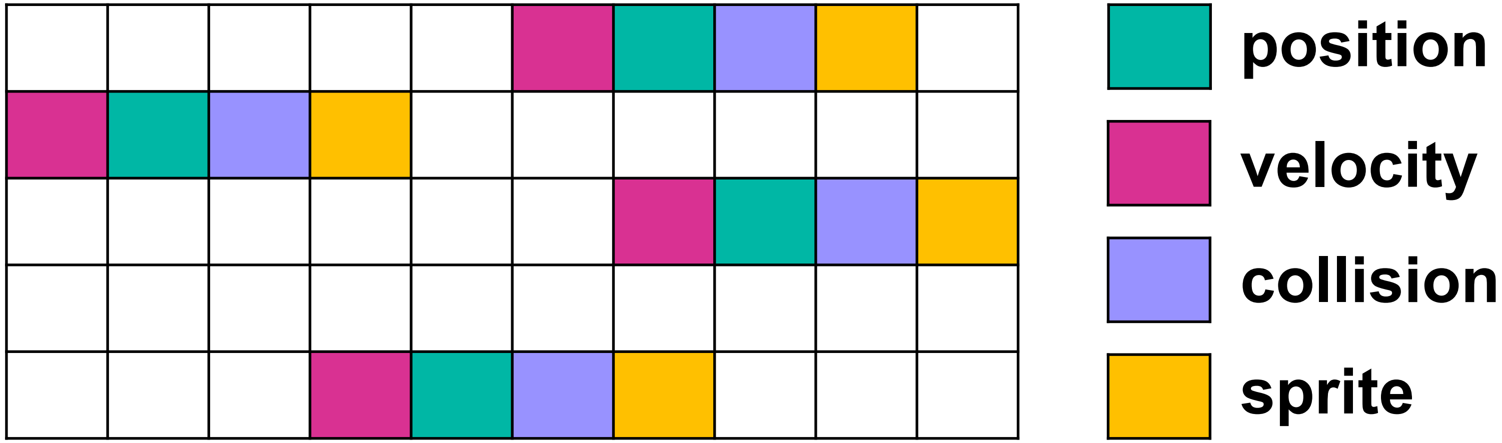
ECS = containers of components?

Memory & ECS

Where do we store our Components?

- Inside Entities?

Update loop has to access non-contiguous memory repeatedly!



Memory Blocks

Slow memory access!

iter

ECS = std::map?

- **Associate components to entities**
- **Dynamic!**
- **Fast?**

Task	**(hash) map**
Dynamically add/remove a component	
Check if entity has N components	
Get component of type X of entity	
Iterate over all components of type X	
Iterate over all entities with component X	

insert, emplace, erase

count

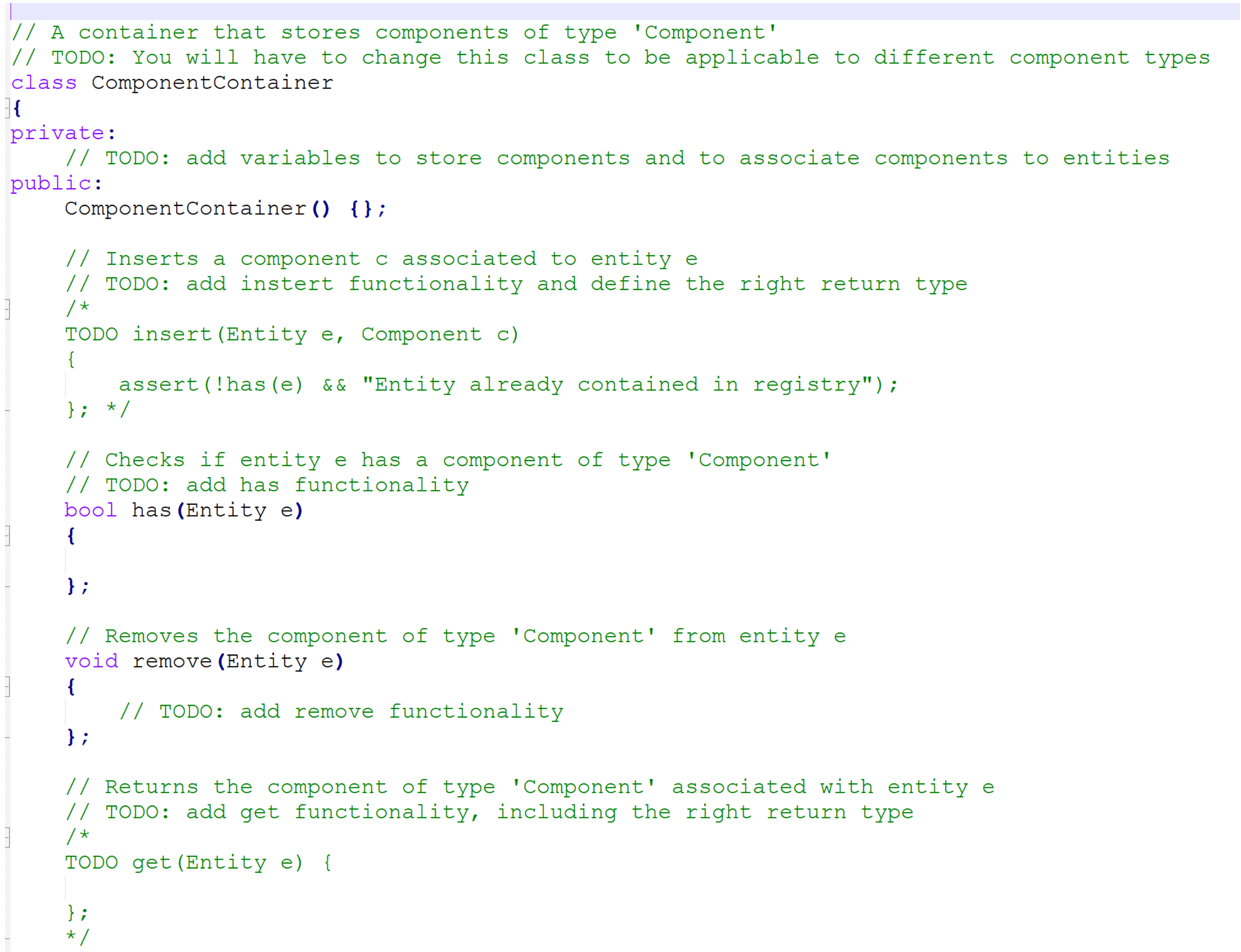
find

begin() iter->second

begin() iter->first

Try std::map out for A0

We will release a template

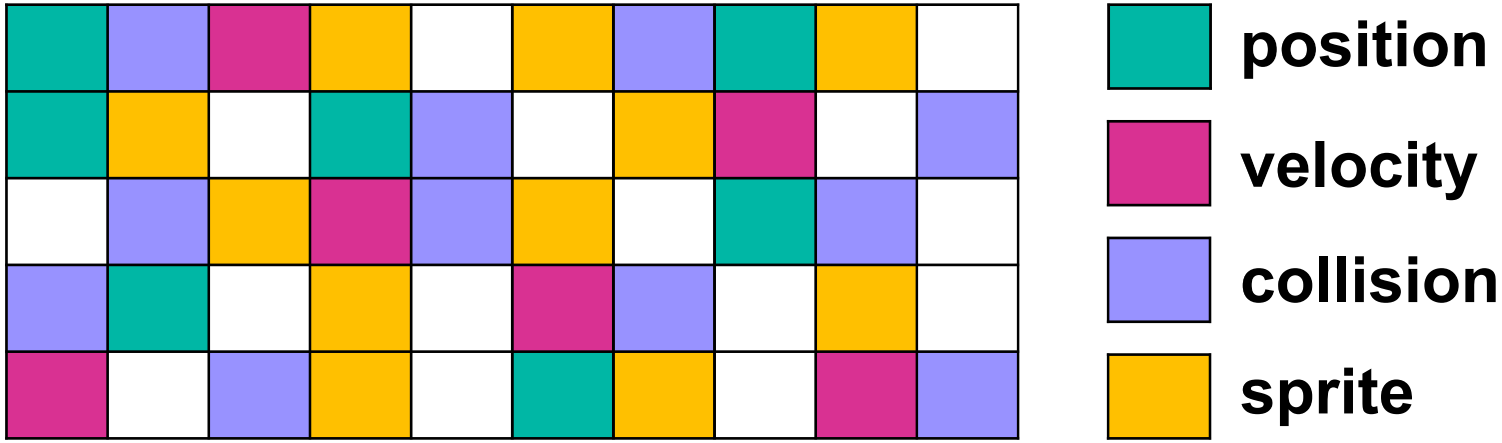


```
// A container that stores components of type 'Component'
// TODO: You will have to change this class to be applicable to different component types
class ComponentContainer
{
private:
    // TODO: add variables to store components and to associate components to entities
public:
    ComponentContainer() {};

    // Inserts a component c associated to entity e
    // TODO: add instert functionality and define the right return type
    /*
    TODO insert(Entity e, Component c)
    {
        assert(!has(e) && "Entity already contained in registry");
    }; */

    // Checks if entity e has a component of type 'Component'
    // TODO: add has functionality
    bool has(Entity e)
    {

    };

    // Removes the component of type 'Component' from entity e
    void remove(Entity e)
    {
        // TODO: add remove functionality
    };

    // Returns the component of type 'Component' associated with entity e
    // TODO: add get functionality, including the right return type
    /*
    TODO get(Entity e) {

    };
    */
```

Memory & ECS

Where do we store our Components?

- In a map?
  - *It has all the functionalities*



Memory Blocks

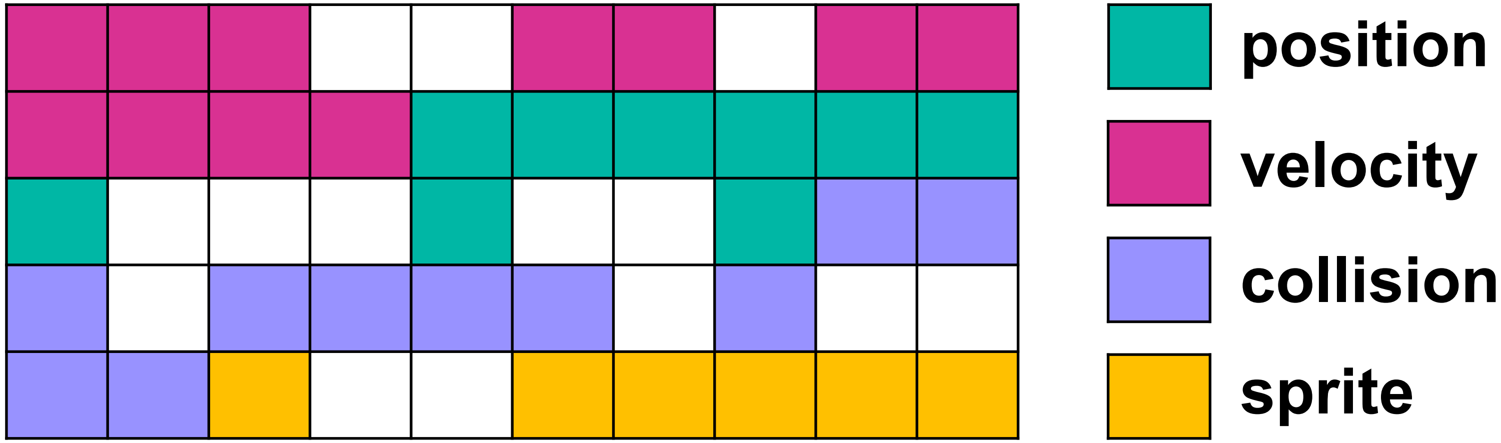
Update loop has to access non-contiguous memory repeatedly!

Slow memory access!

Memory & ECS

Where do we store our Components?

- Array with holes?

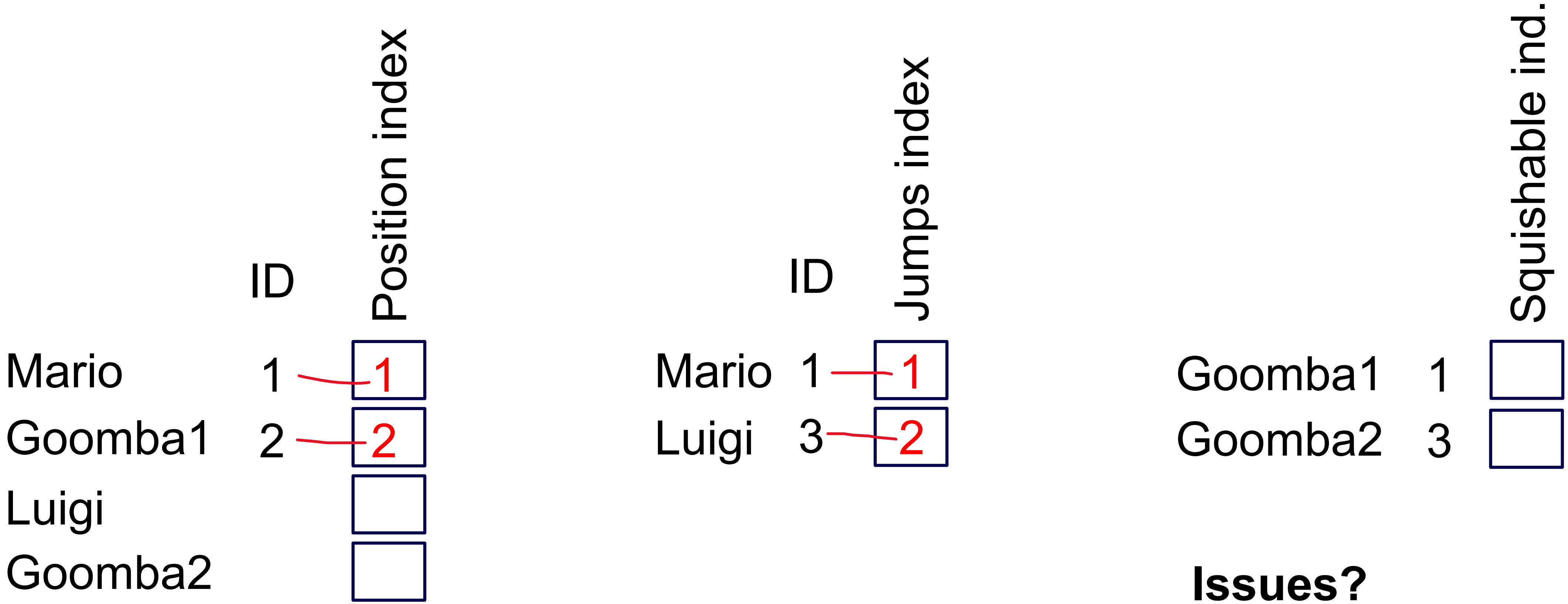


Memory Blocks

Better cache utilization!

Not memory efficient!

Map + Dense Component Vectors (entity ID to component ~~address~~ index)



	ID	Position index
Mario	1	1
Goomba1	2	2
Luigi		
Goomba2		

	ID	Jumps index
Mario	1	1
Luigi	3	2

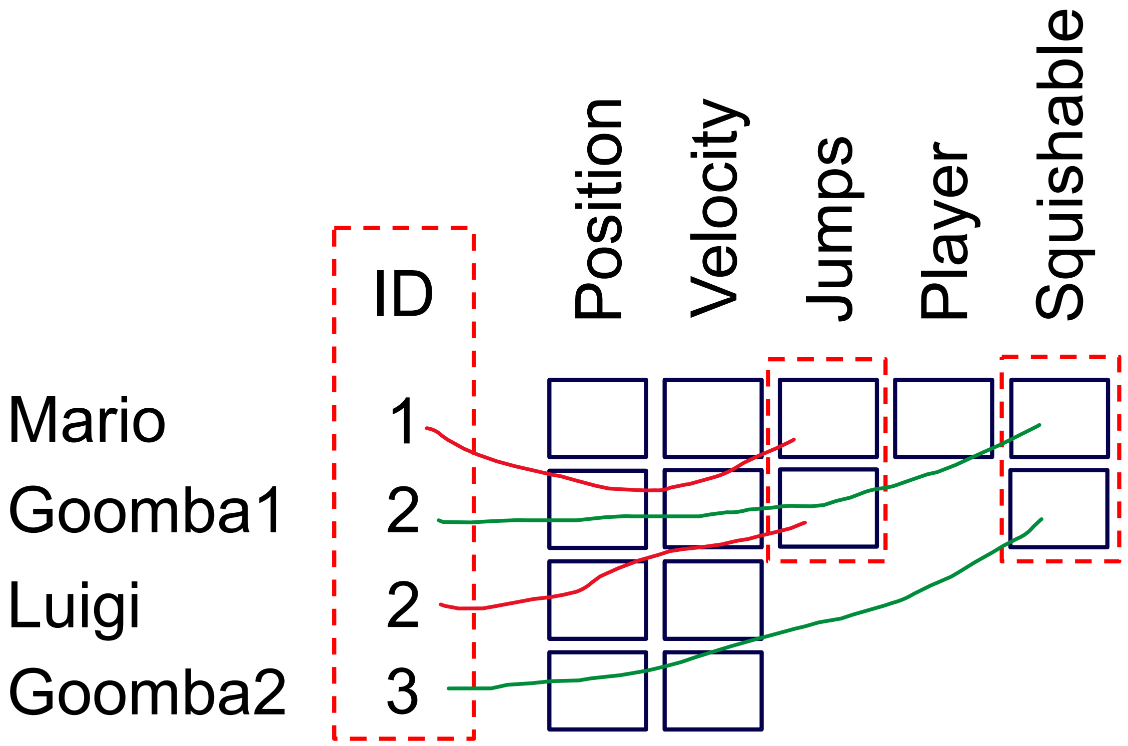
		Squishable ind.
Goomba1	1	
Goomba2	3	

Issues?

Concept: Combine dense vectors with a map

Implementation: std::vector<Component>; std::map<Entity,unsigned int>

Map + Dense Vector (different visualization)



ID

Position

Velocity

Jumps

Player

Squishable

Mario 1

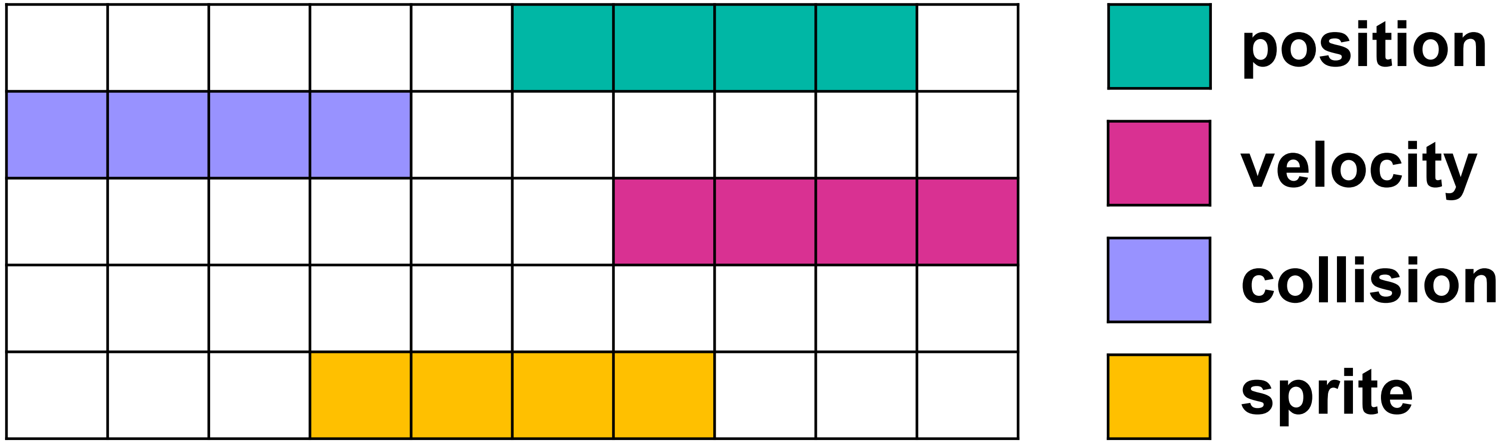
Goomba1 2

Luigi 2

Goomba2 3

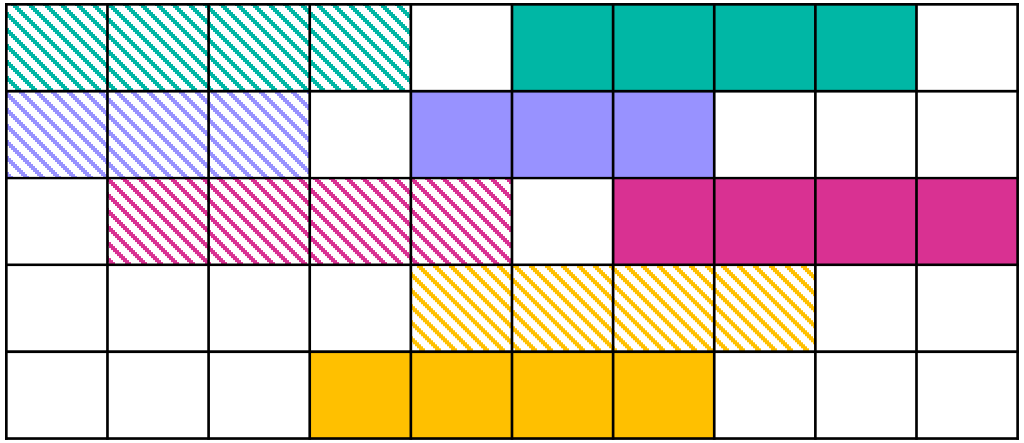
Cache is Key

- Each Component type has a statically allocated array
- Minimizes costly cache misses
  - *Keeps components we access around the same time close to each other*



Memory Blocks

Map + Component Vectors + Entity Vector Cache is Key

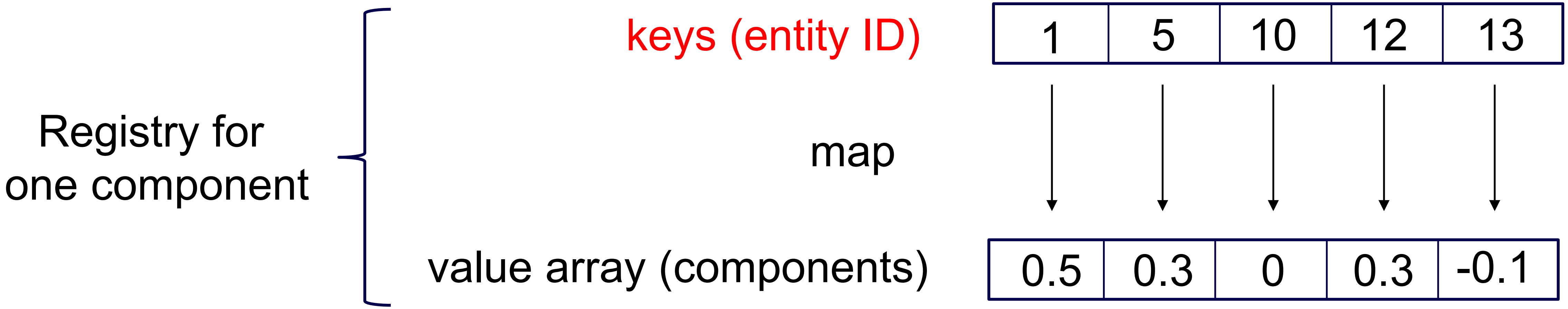


Memory Blocks

- 🟩 position
- 🟥 velocity
- 🟪 collision
- 🟨 sprite
- ▨ position entity IDs
- ▨ velocity entity IDs
- ▨ collision entity IDs
- ▨ sprite entity IDs

Update loop accesses contiguous memory IDEAL!

Map + Component Vector + Entity Vector



Concept: Add a dense vector of entities to facilitate quick iteration over entities
Implementation: std::vector<Entities>; std::vector<Component>; std::map<Entity,unsigned int>

Easy to iterate over all velocity components that belong to an entity with a position

```
for(int entity : velocity_entities) // using the entities array
    if (position_entity_map.has(entity)) // using the map
        position_entity_map.get(entity) += velocity_entity_map.get(entity); // using component array
```

Faster iteration via entity and component array

Accessing the velocity map (reg_velocity.map) is an unnecessary indirection

```
for(int entity : velocity_entities) // efficient
    if (position_entity_map.has(entity)) // inefficient lookup
            position_entity_map.get(entity) += velocity_entity_map.get(entity); // 2x inefficient lookup
```

We can access the velocity components in linear fashion

```
for(int vel_i = 0; vel_i < velocity_entities.size(); vel_i++) // efficient
   Entity entity : velocity_entities[vel_i]; // efficient
   int pos_i = position_entity_map.getIndex(entity); // inefficient lookup
   if (pos_i)
            position_components[pos_i] += reg_velocity_components[vel_i]; // efficient
```

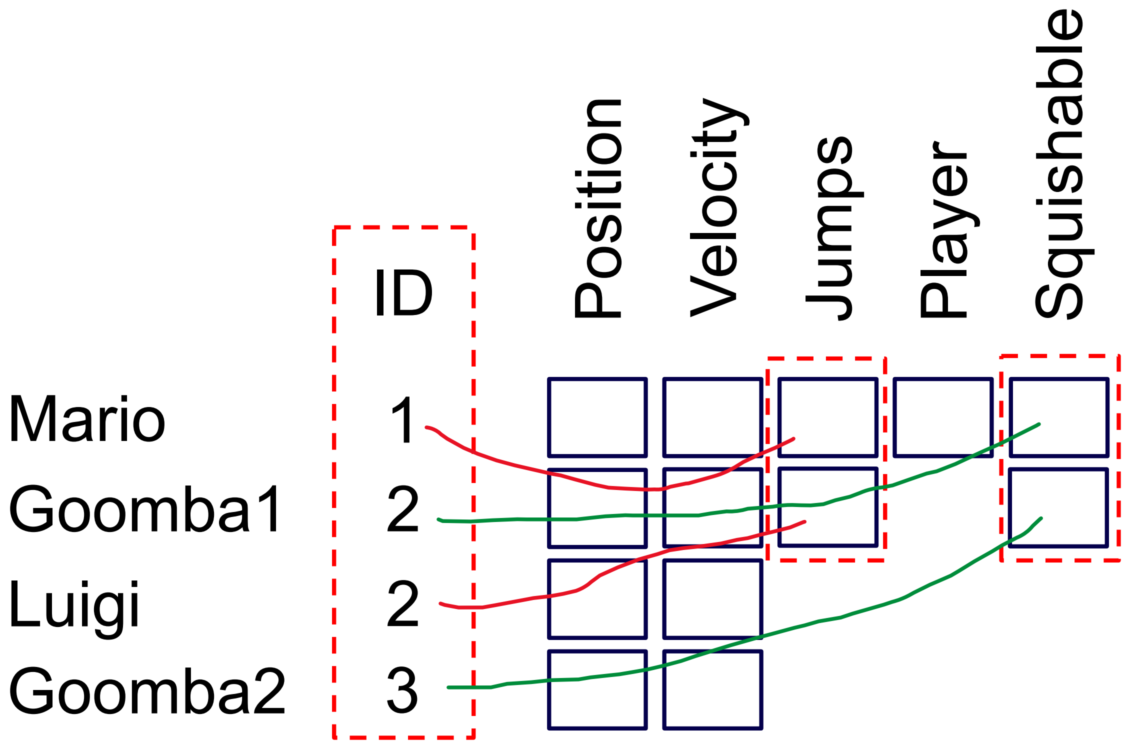
ECS goals: fast & dynamic

- **Associate components to entities**
- **Fast & dynami**

Task	(hash) map	bitset	Comp. vec	Entity vec
Dynamically add/rem. a comp.		-		
Check if entity has N components			-	-
Get component X of entity		-	-	-
Iterate over comp. of type X		-		
Iterate over ent. with comp. X		-		

If you want to take a deep dive…

Self-study: A special map approach



	ID
Mario	1
Goomba1	2
Luigi	2
Goomba2	3

Position

Velocity

Jumps

Player

Squishable

Self-study: The ‘Sparse Set’

	ID	Index Pos	Index Vel	Index Jump	Index Player	Index Squish	Position	Velocity	Jumps	Player	Squishable
Mario	1	□	□	1	1		□	□	□	□	□
Goomba1	2	□	□			1	□	□	□		□
Luigi	3	□	□	2			□	□			
Goomba2	4	□	□			2	□	□			

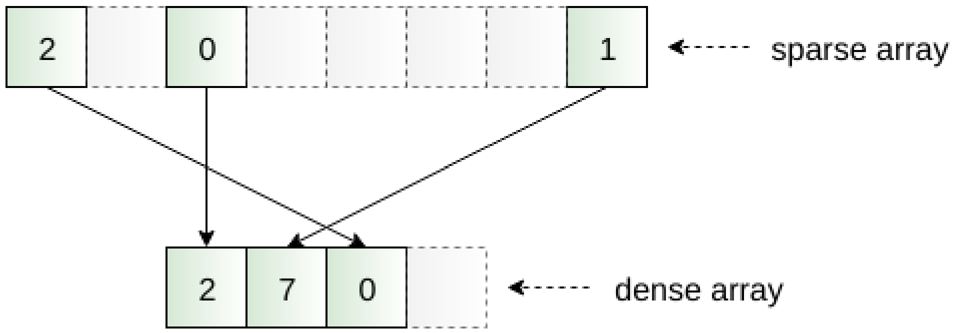
Issues?

Concept: Sparse array + dense array

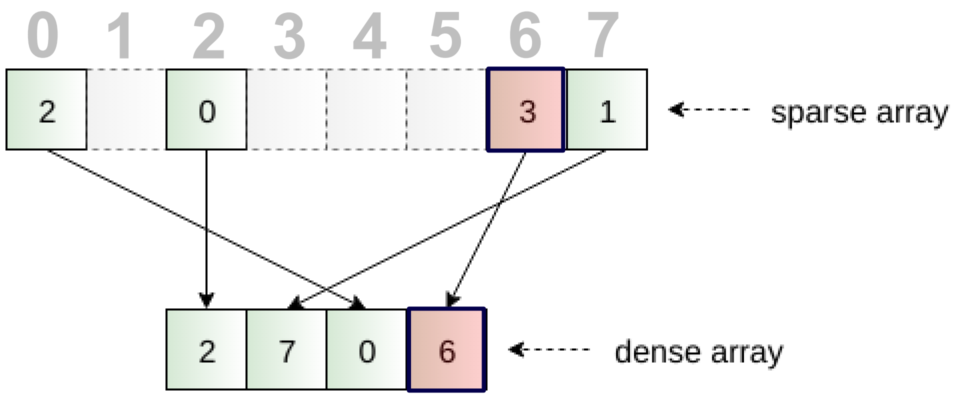
Implementation: std:vector<Entity> entities; std:vector<unsigned int> indices; std:vector<Components> components;

Self-study: Faster Lookup with Sparse Sets

Lookup:



Insert:



The map lookup (map.get(entity)) is costly

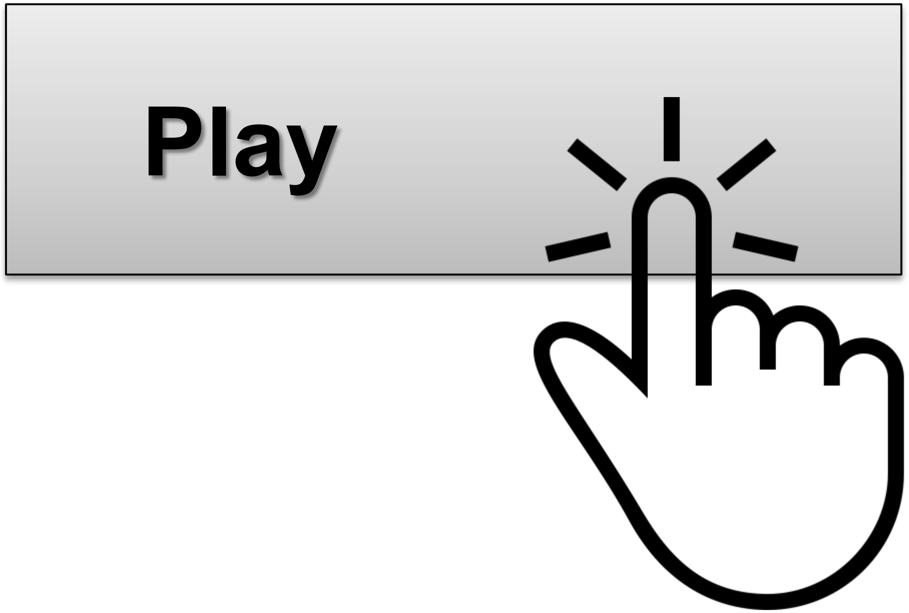
- A hashmap is O(1), but that 1 is big

Sparse set:

- An array as large as the number of entities in the game
  - Crazy waste of memory?!
  - 32 bit integer -> ???
  - a sparsely filled array
- A small dense array of all entities in sequence (as before)
- Extremely fast lookup, insert, & clear

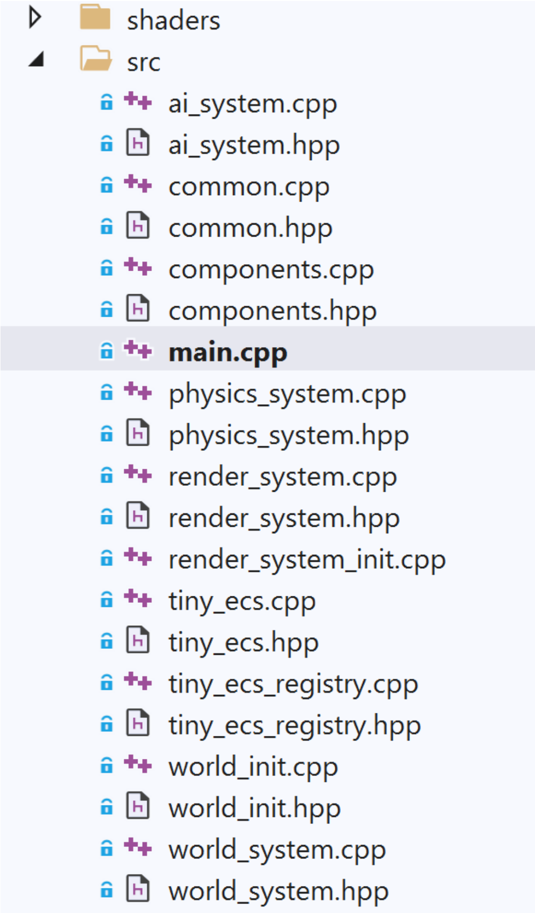
[https://skypjack.github.io/2020-08-02-ecs-baf-part-9/]

Game Programming Basics

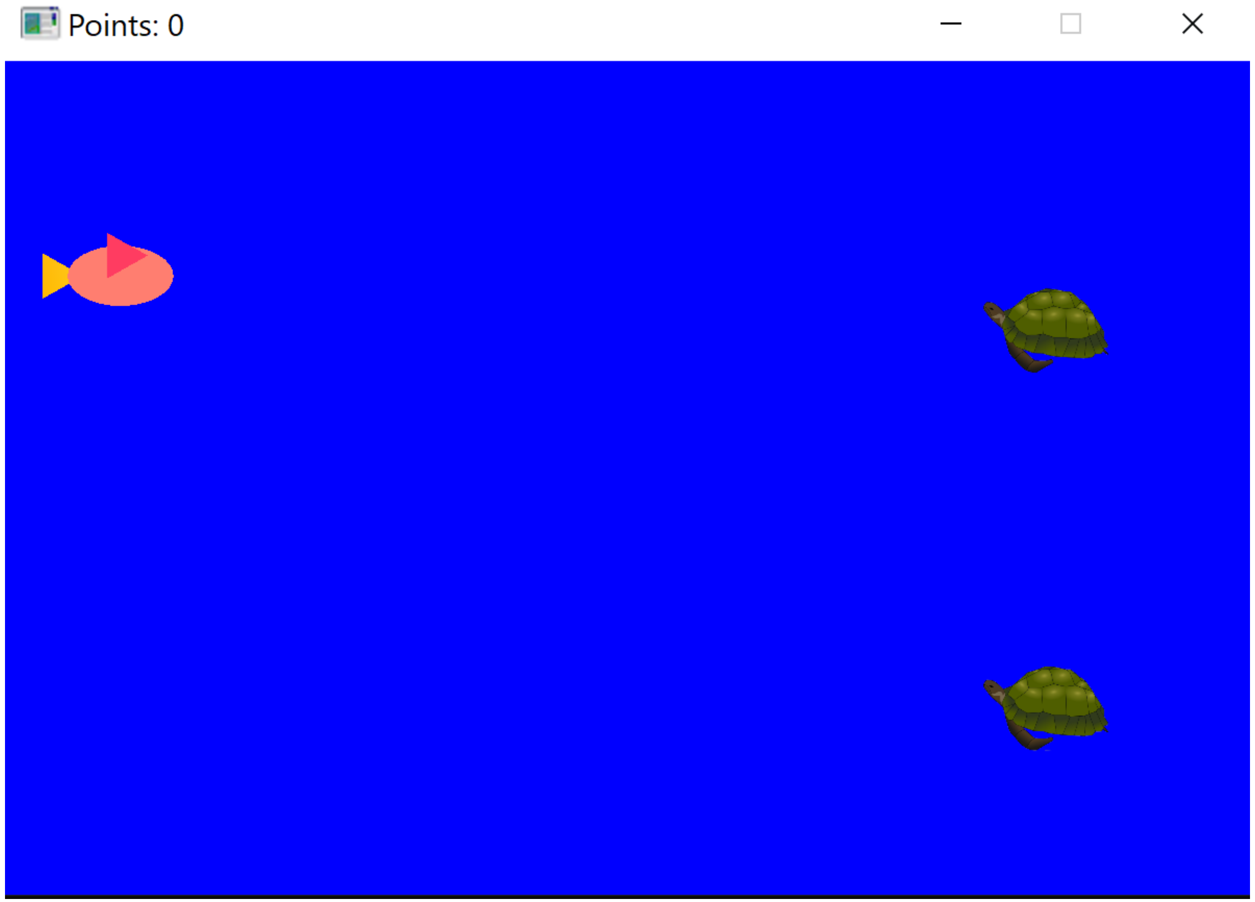


Assignments

Template framework

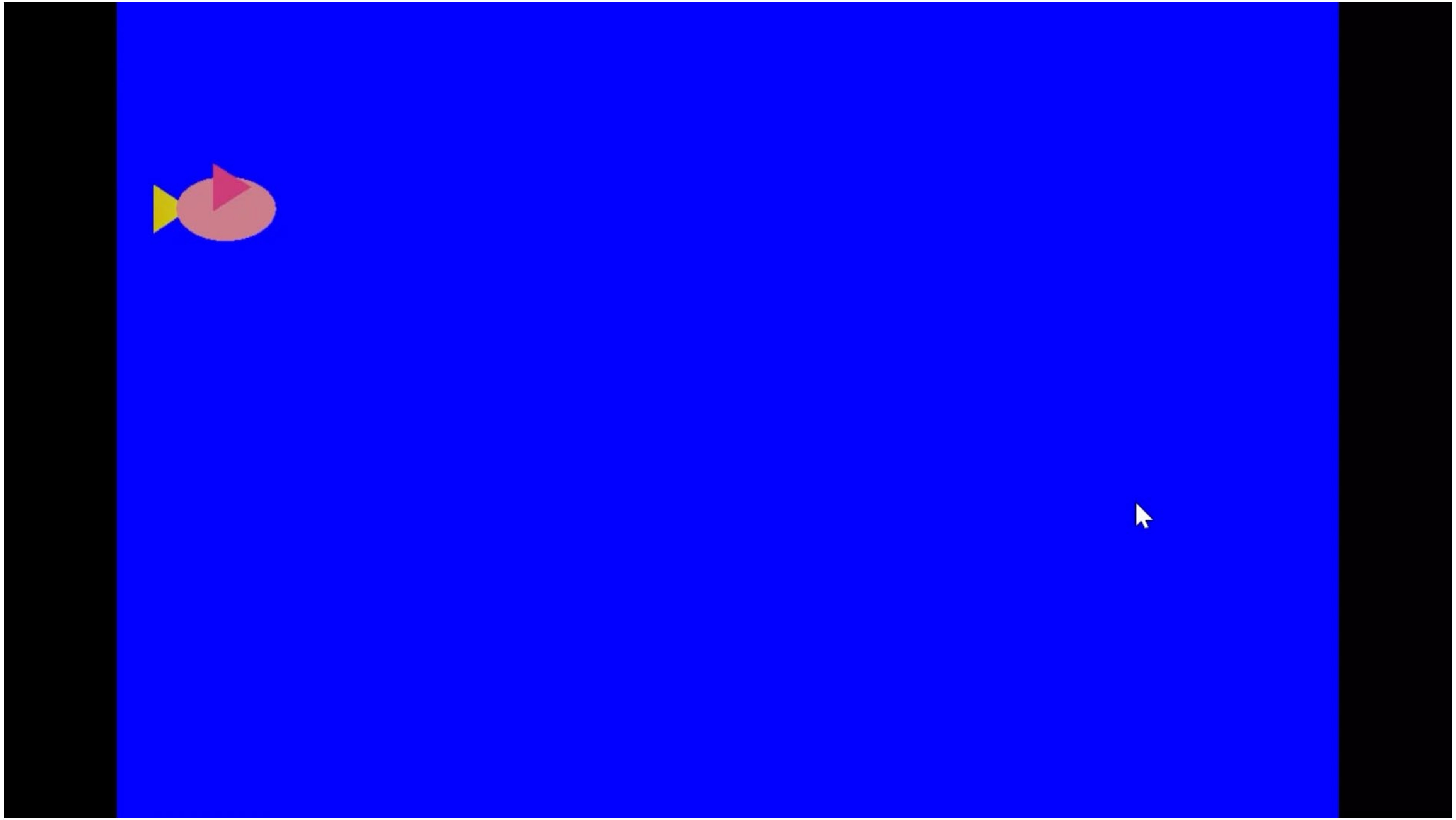


- shaders
- src
  - ai_system.cpp
  - ai_system.hpp
  - common.cpp
  - common.hpp
  - components.cpp
  - components.hpp
  - **main.cpp**
  - physics_system.cpp
  - physics_system.hpp
  - render_system.cpp
  - render_system.hpp
  - render_system_init.cpp
  - tiny_ecs.cpp
  - tiny_ecs.hpp
  - tiny_ecs_registry.cpp
  - tiny_ecs_registry.hpp
  - world_init.cpp
  - world_init.hpp
  - world_system.cpp
  - world_system.hpp

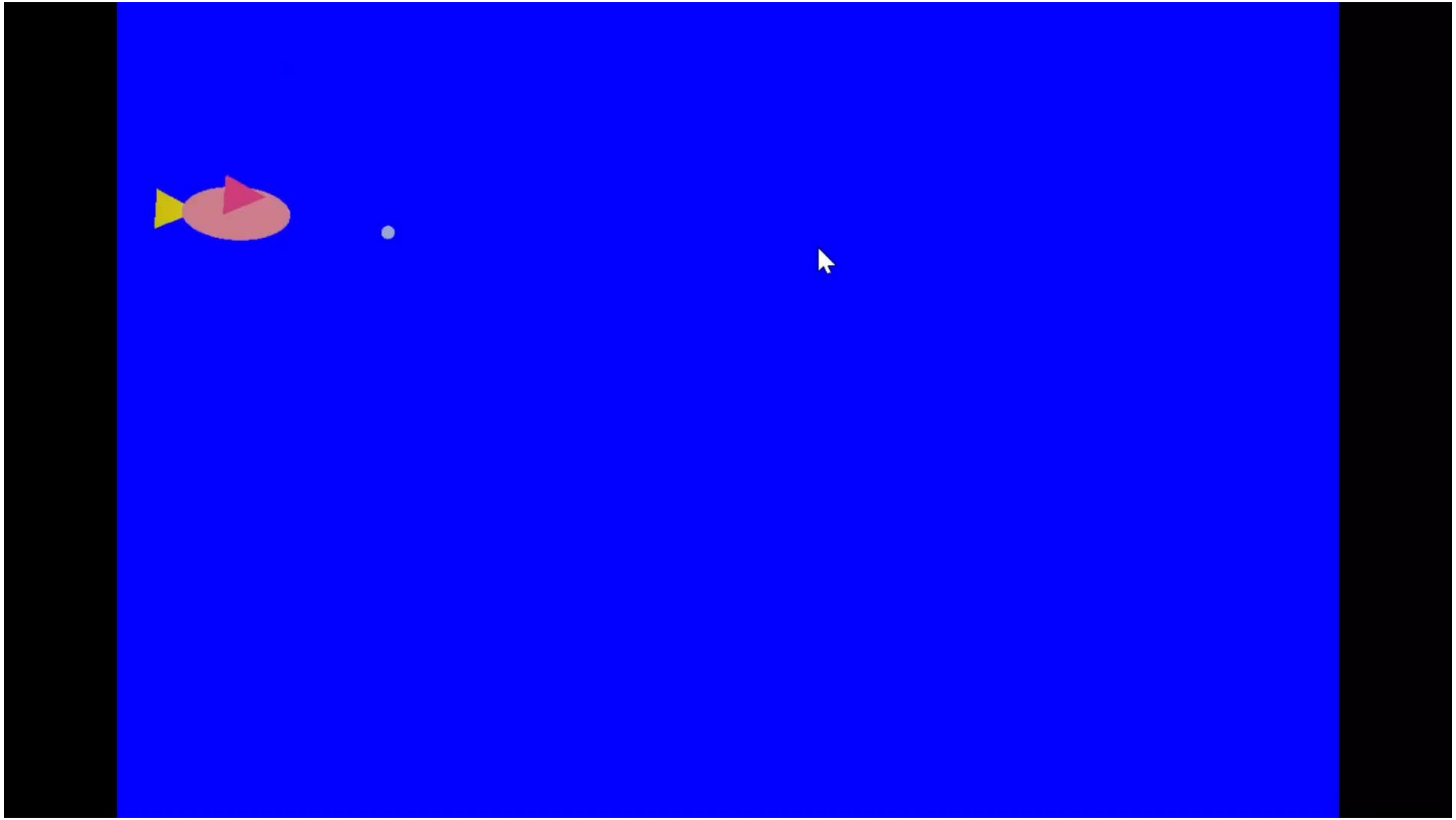


Points: 0

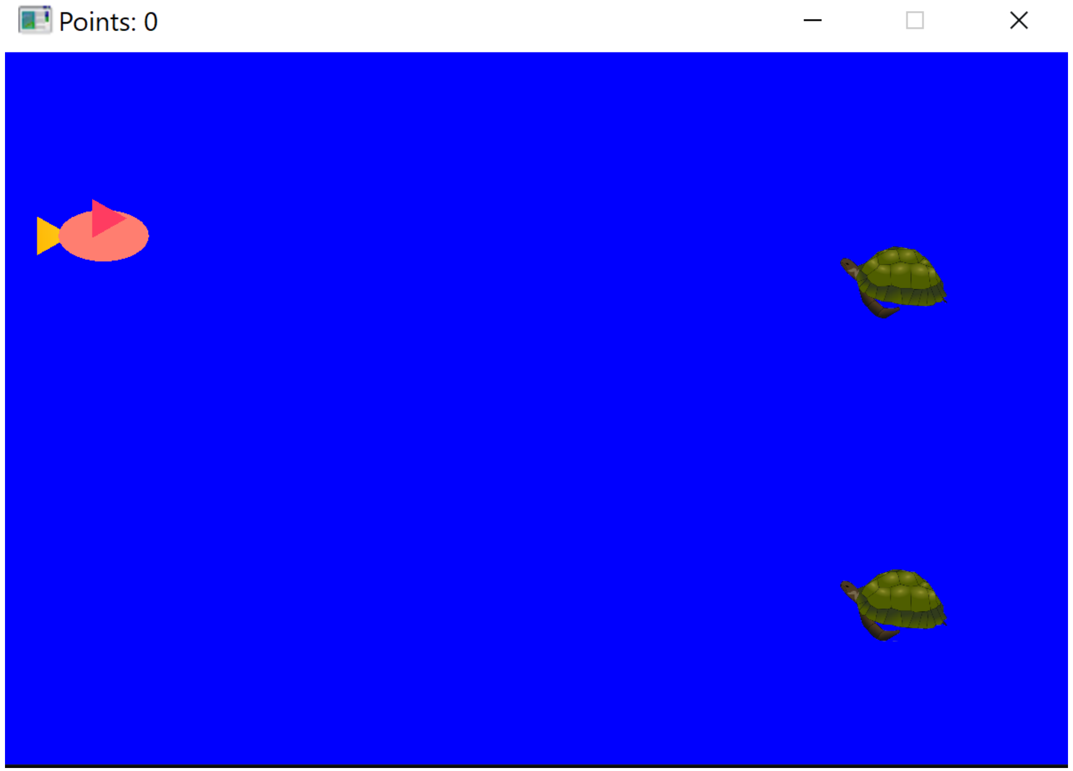
A1 – Game Graphics

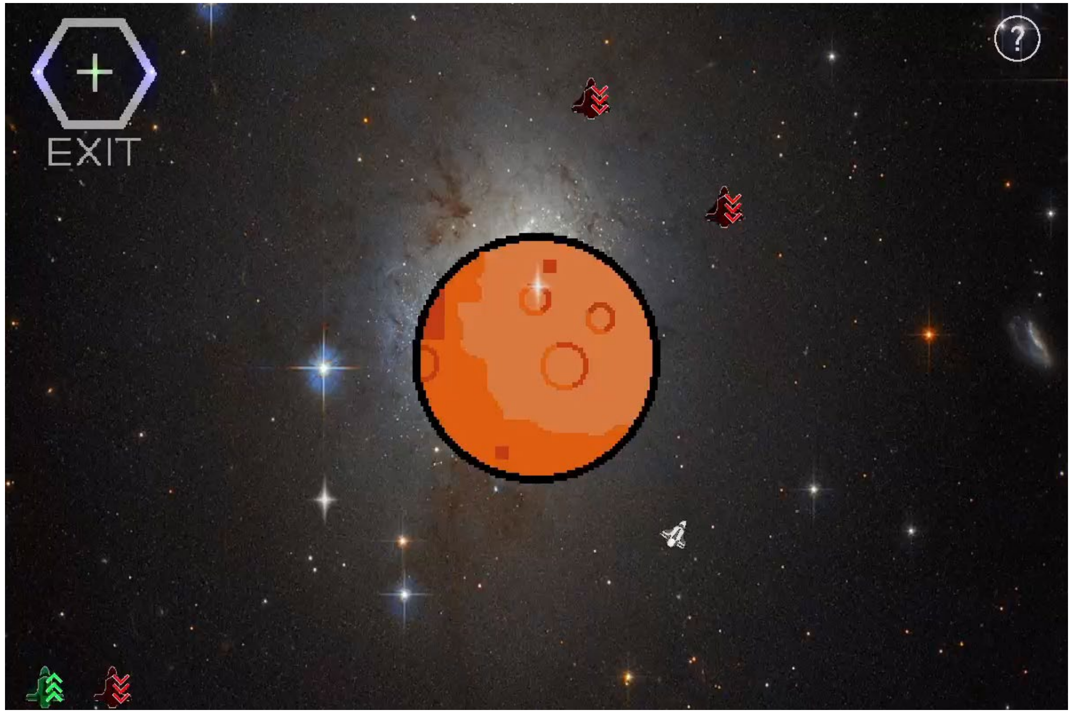


A2 – Animation and Physics



Your project


Points: 0



Procedural Programming

Sequential control flow

- program performs a sequence of tasks & terminates
- good for physical simulation
- maintains consistent rendering frame rate

- difficult to model a long order of events

Event-Driven Programming

No main loop under your control

- vs. procedural

Control flow through event callbacks

- redraw the window now
- key was pressed -> react
- mouse moved -> react

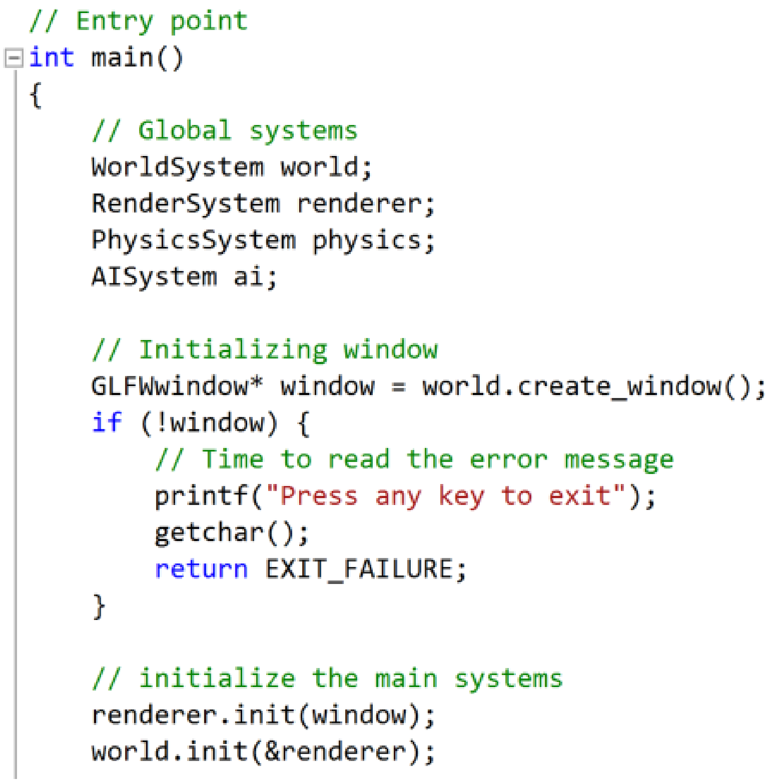
Callback functions called from main loop when events occur

- mouse/keyboard
- ensures temporal order
- prevents concurrency

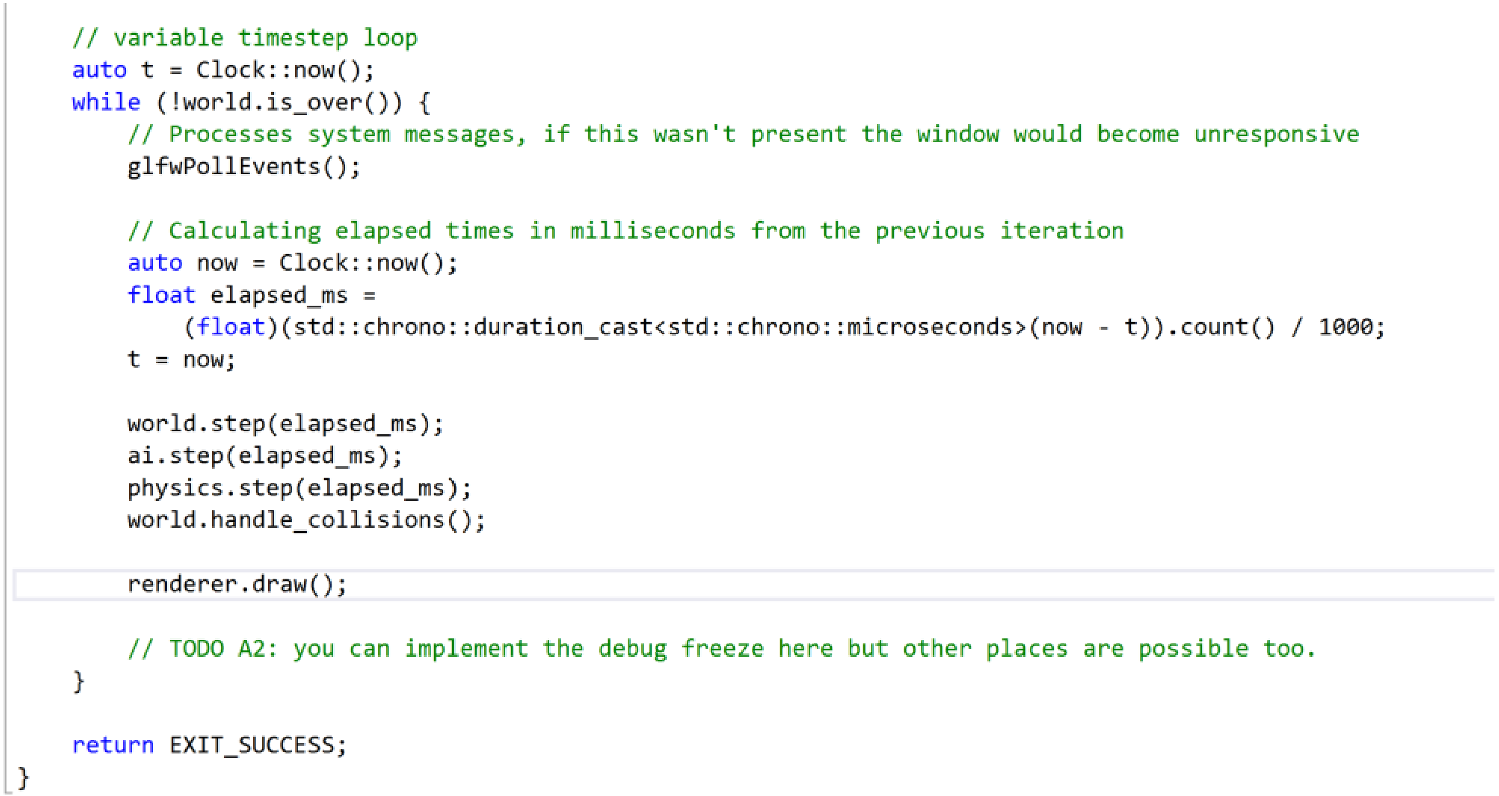
Minimal Main (openGL)

```
int main(int argc, char* argv[]) {
  if (!world.init(..)){
    return EXIT_FAILURE;
  }
  while (!world.is_over()) {
      glfwPollEvents(); // process events
      world.update(); // update game state based on events + timer
      world.draw(); // render
  }
  world.destroy();
  return EXIT_SUCCESS;
}
```

Our game loop (A1-A3 Template, main.cpp)



```cpp
// Entry point
int main()
{
    // Global systems
    WorldSystem world;
    RenderSystem renderer;
    PhysicsSystem physics;
    AISystem ai;

    // Initializing window
    GLFWwindow* window = world.create_window();
    if (!window) {
        // Time to read the error message
        printf("Press any key to exit");
        getchar();
        return EXIT_FAILURE;
    }

    // initialize the main systems
    renderer.init(window);
    world.init(&renderer);
```



```cpp
    // variable timestep loop
    auto t = Clock::now();
    while (!world.is_over()) {
        // Processes system messages, if this wasn't present the window would become unresponsive
        glfwPollEvents();

        // Calculating elapsed times in milliseconds from the previous iteration
        auto now = Clock::now();
        float elapsed_ms =
            (float)(std::chrono::duration_cast<std::chrono::microseconds>(now - t)).count() / 1000;
        t = now;

        world.step(elapsed_ms);
        ai.step(elapsed_ms);
        physics.step(elapsed_ms);
        world.handle_collisions();

        renderer.draw();

        // TODO A2: you can implement the debug freeze here but other places are possible too.
    }

    return EXIT_SUCCESS;
}
```

openGL

- Low-level graphics API
- C Interface accessed from C++
- Shaders – graphics
  - *A LOT more details later*

Even Callbacks

Set at start – in our template in world.init()

```
auto key_redirect = [](GLFWwindow* wnd, int _0, int _1, int _2, int _3) {
    ((World*)glfwGetWindowUserPointer(wnd))->on_key(wnd, _0, _1, _2, _3); };
auto cursor_pos_redirect = [](GLFWwindow* wnd, double _0, double _1) {
    ((World*)glfwGetWindowUserPointer(wnd))->on_mouse_move(wnd, _0, _1); };
glfwSetKeyCallback(m_window, key_redirect);
glfwSetCursorPosCallback(m_window, cursor_pos_redirect);
```

Another example would be a mouse click (same format)

Callback Actions

```
void World::on_key(GLFWwindow*, int key, int, int action, int mod){
if (action == GLFW_RELEASE && key == GLFW_KEY_R){
    …
}
if (action == GLFW_RELEASE && (mod & GLFW_MOD_SHIFT) &&  key ==
GLFW_KEY_COMMA){
  …
}
void World::on_mouse_move(GLFWwindow* window, double xpos, double
ypos){
}
```