G54GAM - Games

- Software architecture of a game

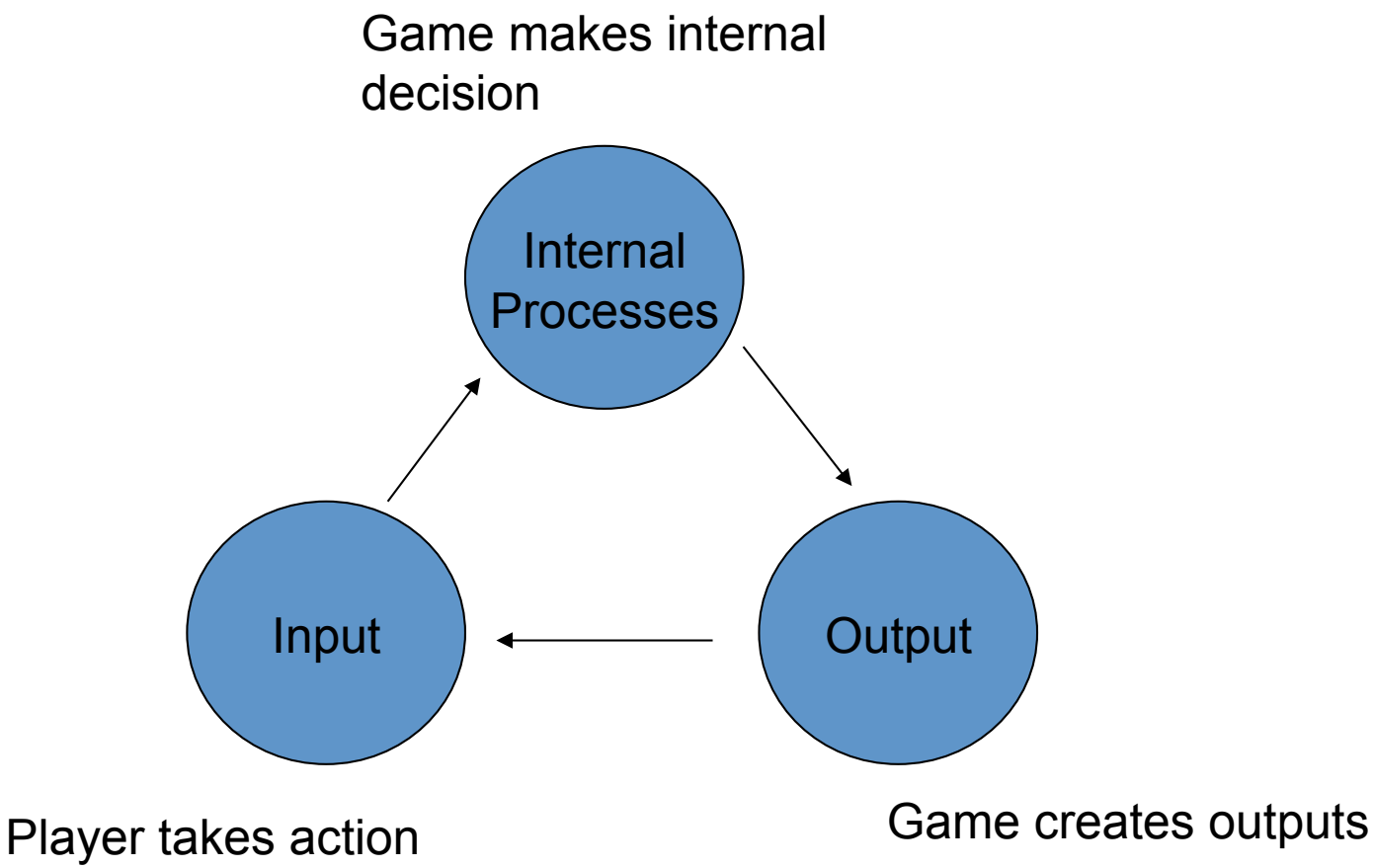
Coursework

- Coursework 2 and 3 due 18th May
- Design and implement **prototype** game
  - Write a game design document
  - Make a working prototype of a game
  - Make use of the lab sessions to discuss your ideas
- Coursework document on the website
  - http://www.cs.nott.ac.uk/~mdf/teaching_G54GAM.html

Now what?

- Now that we have our game design, how do we go about building it?
  - Complex interactive system
- We need to plan it otherwise it becomes a mess
  - Difficult to understand
  - Difficult to maintain
  - Difficult to extend
- “Games must be designed, but computers must be programmed”
  - Still mainly native – C, C++
  - However we can write a game in any language we wish

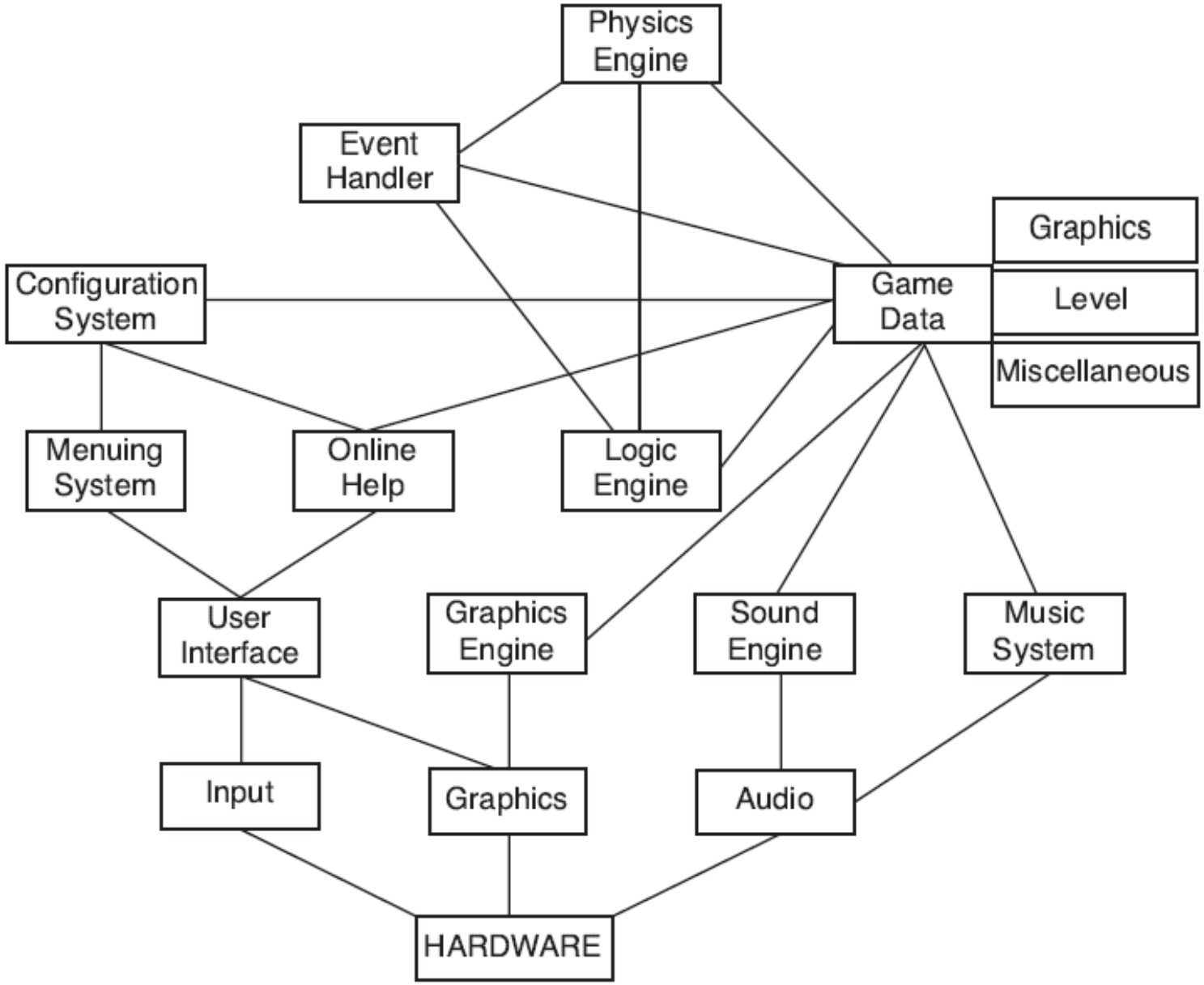
Interactive System



How do we put it all together?

- Inputs
  - Mouse, keyboard, controller
- Internal Processes
  - Evolving Game State
  - Objects, Rules, Procedures…
- Outputs
  - Graphics
  - Sound
  - User Interface

What are common game system components?


Physics Engine
Event Handler
Graphics
Configuration System
Game Data
Level
Miscellaneous
Menuing System
Online Help
Logic Engine
User Interface
Graphics Engine
Sound Engine
Music System
Input
Graphics
Audio
HARDWARE

How do we put it all together?

- User interface
  - Configuration and selection
  - Help
  - Input / HUD
- Game Logic
  - Loading
  - Script
  - Physics Engine
  - Artificial Intelligence
  - Events
  - Collisions
  - Network communication
- Outputs
  - Graphics renderer
  - Sound and music

How do we put it all together?

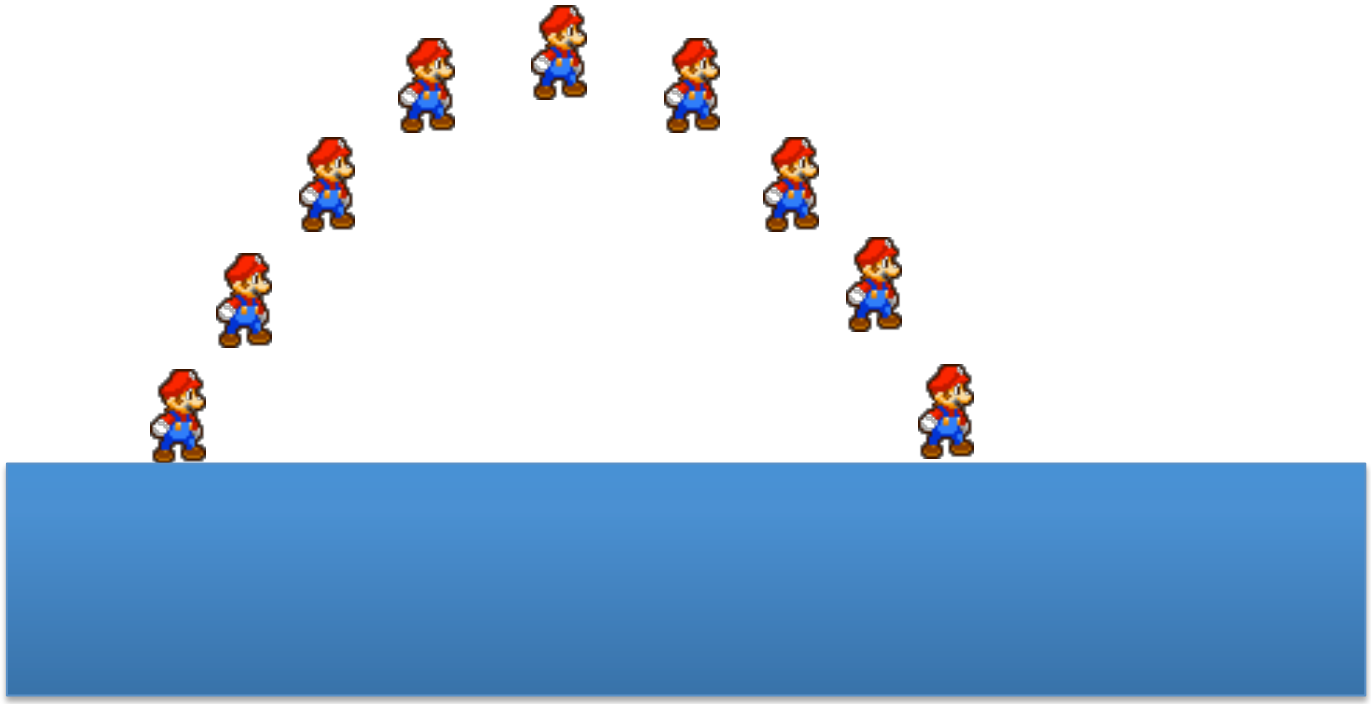
- Game State
  - Position, orientation, velocity of all dynamic entities
  - Behaviour and intentions of AI controlled characters
  - Dynamic, and static attributes of all gameplay entities
    - Scores, health, powerups, damage levels
- All sub-systems in the game are interested in some aspect of the game state.
  - Renderer, Physics, Networking, and Sound systems need to know positions of objects
  - Many systems need to know when a new entity comes into or goes out of existence
  - AI system knows when player is about to be attacked – sound system should play ominous music when this happens

Time and “The Game Loop”

- The “heart beat” of a game
- Performs a series of tasks every **frame**
  - Game state changes over time
  - Each frame is a snapshot of the evolving game state
  - A series of frames are perceived as movement
    - E.g. 60 frames per second
- Run as fast as we can?
  - A smooth game-play experience

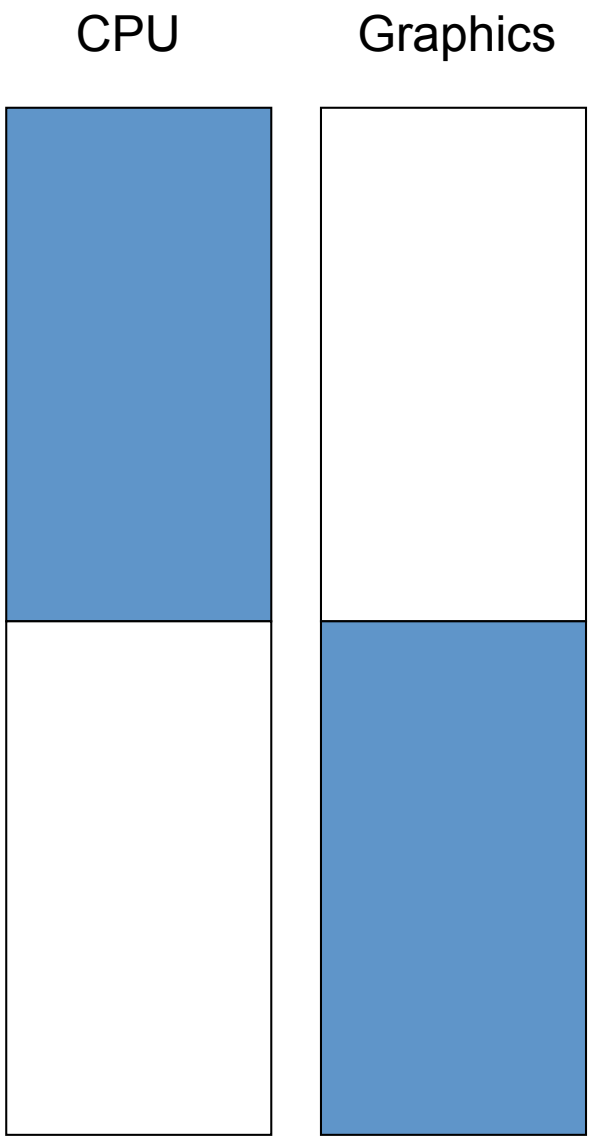
The Game Loop

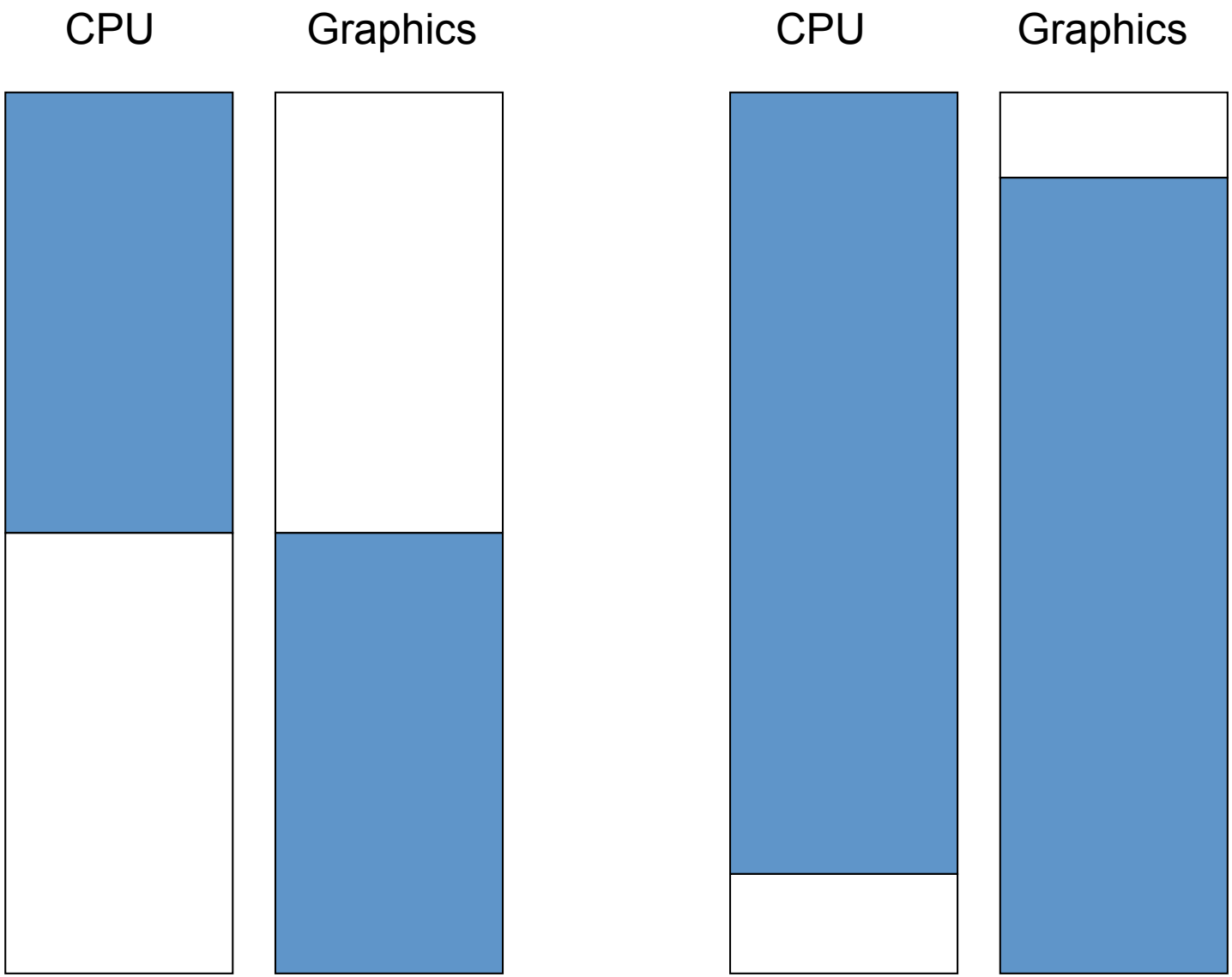
```
start game
while( user doesn't exit )
{
    get user input
    get network messages
    simulate game world
    resolve collisions
    move objects
    draw graphics
    play sounds
}
exit
```



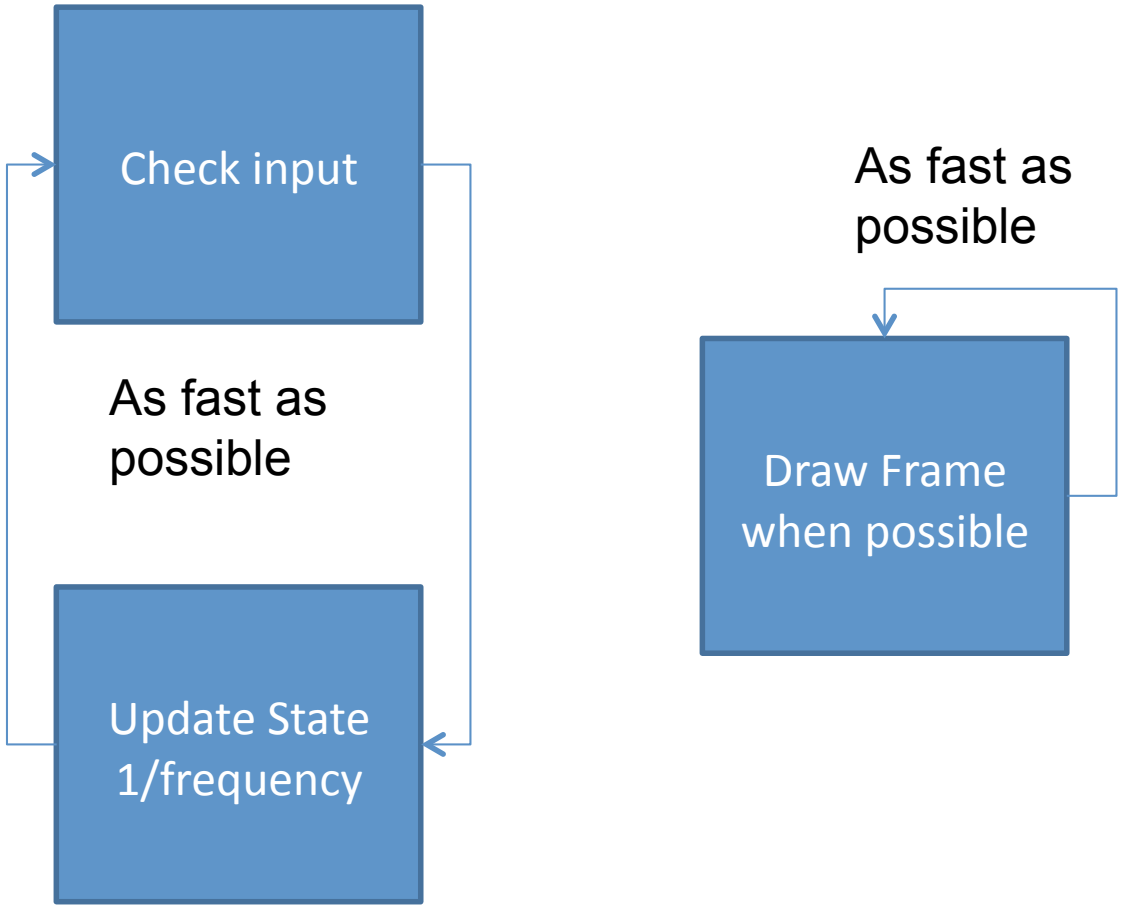
The Game Loop

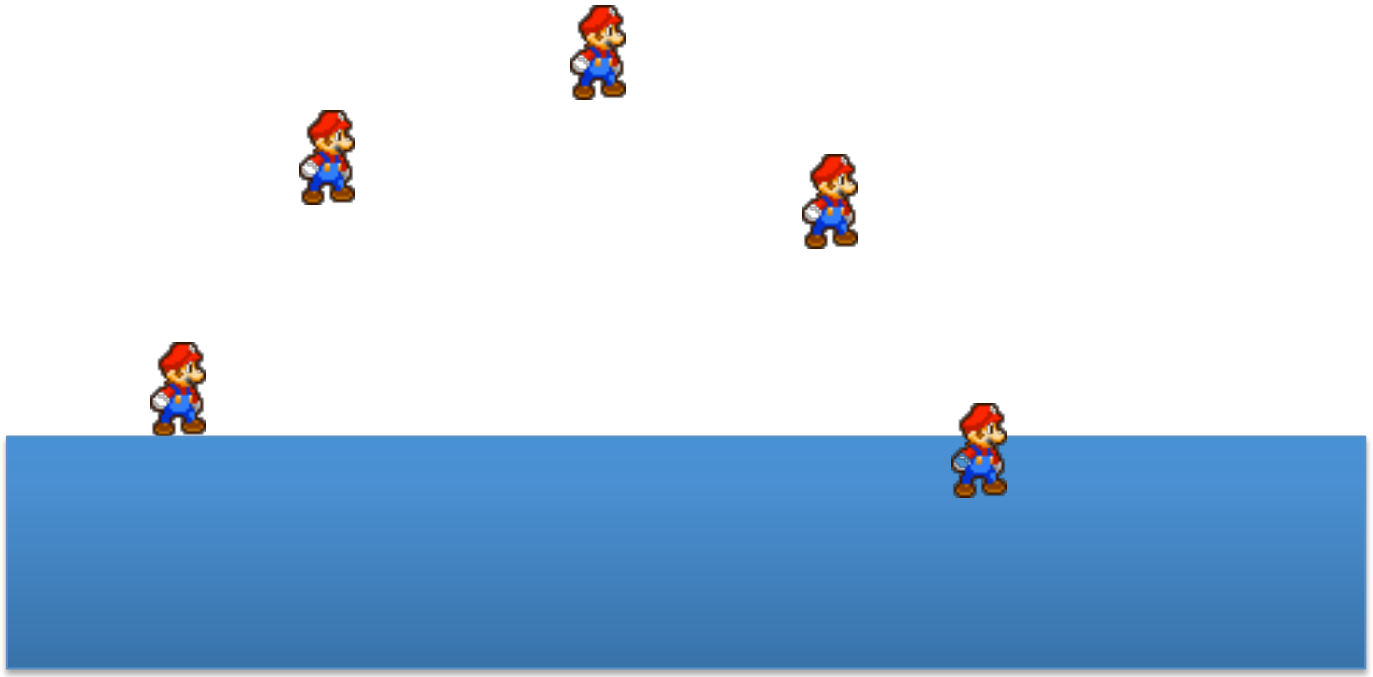
start game
while(user doesn't exit)
{
    how much time has elapsed?
    get user input
    get network messages
    simulate game world(**elapsed time)**
    resolve collisions
    move objects
    draw graphics
    play sounds
    wait (a fixed amount of time)
}
exit


CPU
Graphics


CPU
Graphics
CPU
Graphics

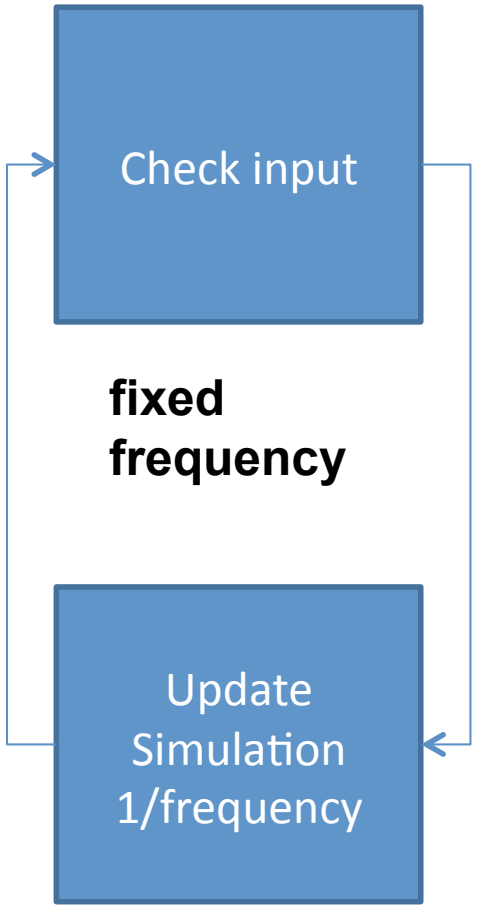
Decoupling

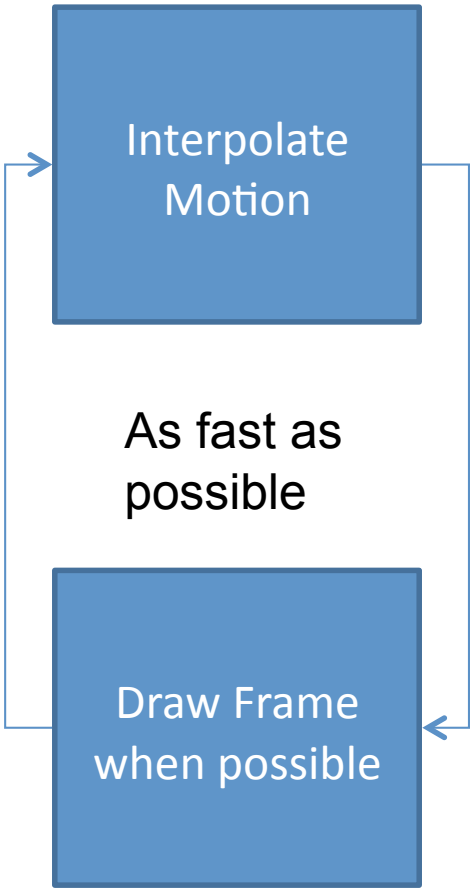

Check input
As fast as possible
Update State 1/frequency
As fast as possible
Draw Frame when possible



Decoupling

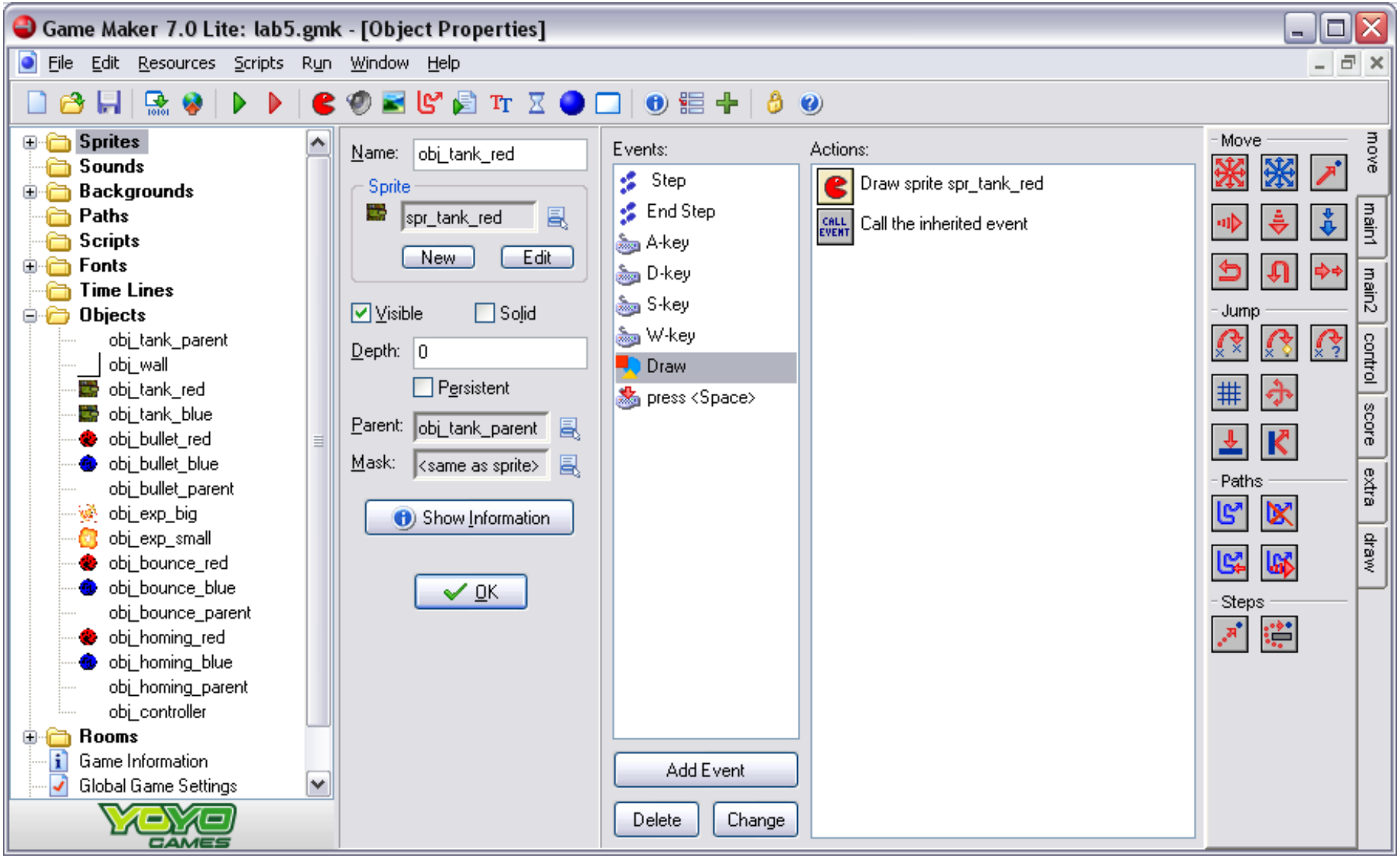
How it plays

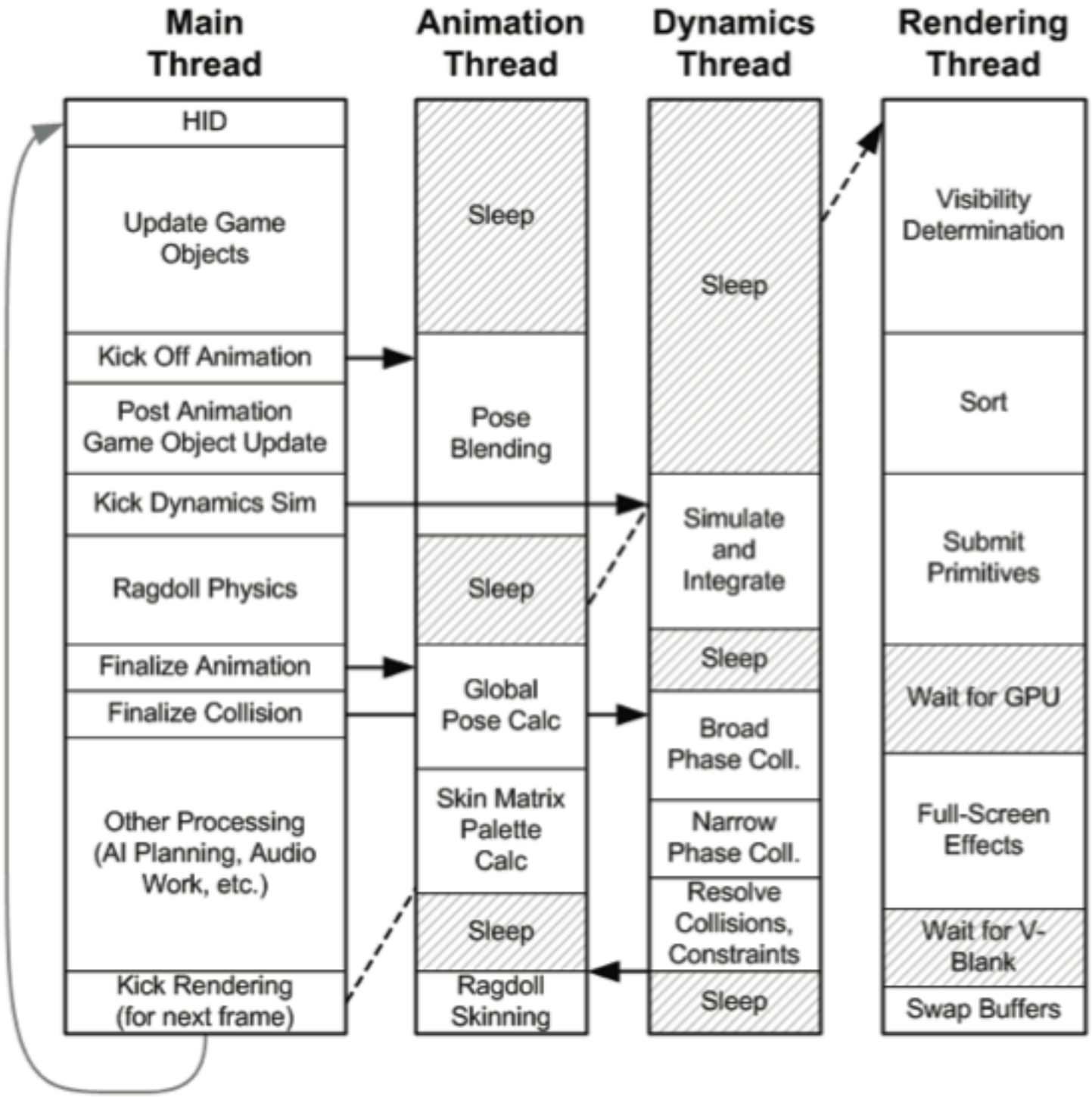




How it looks

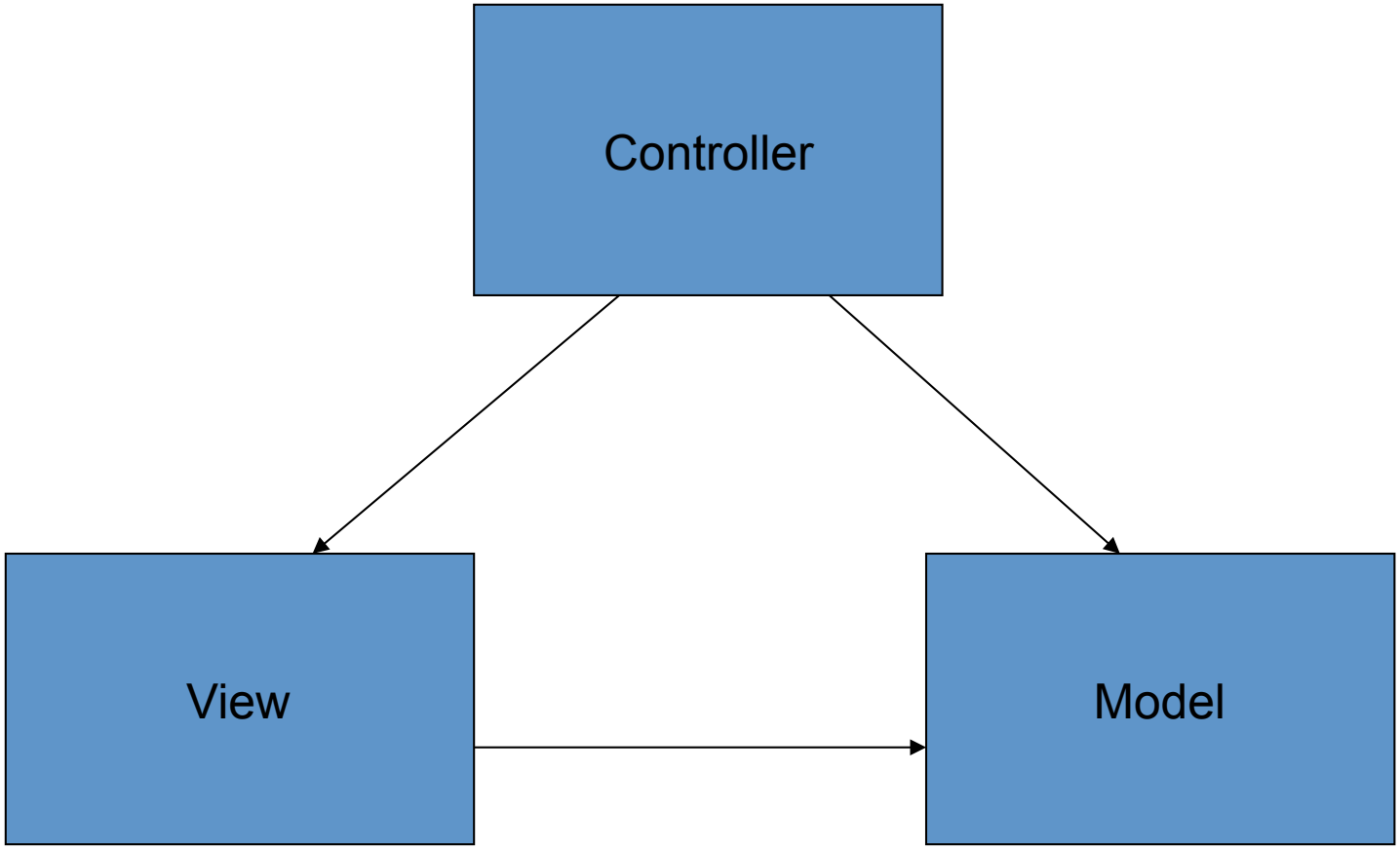

GHOST TEAM


Game Maker 7.0 Lite: lab5.gmk - [Object Properties]
File Edit Resources Scripts Run Window Help
Sprites
Sounds
Backgrounds
Paths
Scripts
Fonts
Time Lines
Objects
obj_tank_parent
obj_wall
obj_tank_red
obj_tank_blue
obj_bullet_red
obj_bullet_blue
obj_bullet_parent
obj_exp_big
obj_exp_small
obj_bounce_red
obj_bounce_blue
obj_bounce_parent
obj_homing_red
obj_homing_blue
obj_homing_parent
obj_controller
Rooms
Game Information
Global Game Settings
Name: obj_tank_red
Sprite
spr_tank_red
New
Edit
Visible
Solid
Depth: 0
Persistent
Parent: obj_tank_parent
Mask: <same as sprite>
Show Information
OK
Events:
Step
End Step
A-key
D-key
S-key
W-key
Draw
press <Space>
Actions:
Draw sprite spr_tank_red
Call the inherited event
Add Event
Delete
Change
Move
Jump
Paths
Steps
move
main1
main2
control
score
extra
draw


Main Thread
Animation Thread
Dynamics Thread
Rendering Thread
HID
Update Game Objects
Kick Off Animation
Post Animation Game Object Update
Kick Dynamics Sim
Ragdoll Physics
Finalize Animation
Finalize Collision
Other Processing (AI Planning, Audio Work, etc.)
Kick Rendering (for next frame)
Sleep
Pose Blending
Sleep
Global Pose Calc
Skin Matrix Palette Calc
Sleep
Ragdoll Skinning
Sleep
Simulate and Integrate
Sleep
Broad Phase Coll.
Narrow Phase Coll.
Resolve Collisions, Constraints
Sleep
Visibility Determination
Sort
Submit Primitives
Wait for GPU
Full-Screen Effects
Wait for V-Blank
Swap Buffers

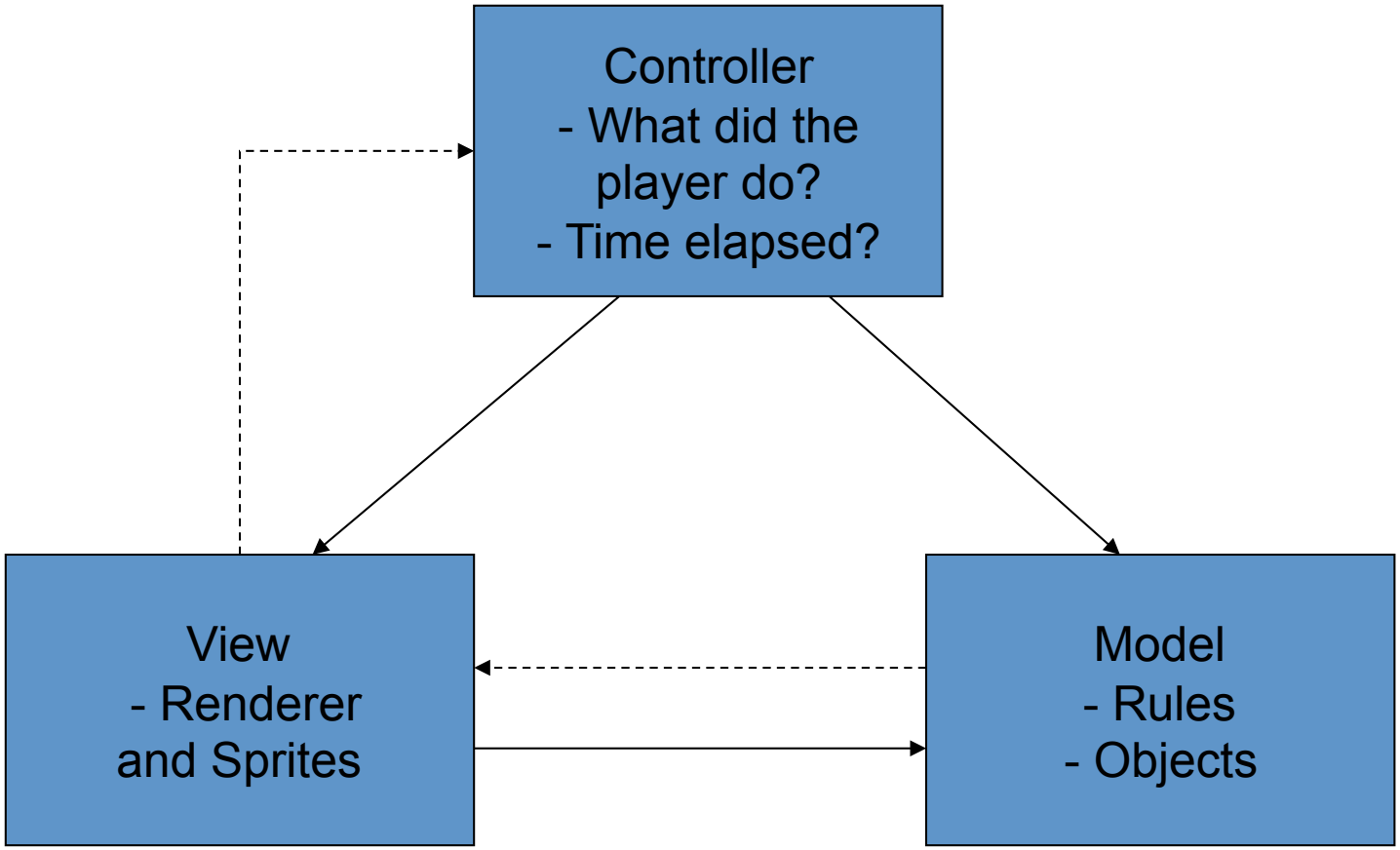
Model-View-Controller

- An architectural design pattern
- Used to isolate logic from user-interface
- Model
  - The information of the application
- View
  - The user interface and display of information
- Controller
  - Manages the communication of information and manipulation of the model


Controller
View
Model

Game MVC Architecture

- Model
  - The state of every game object and entity
  - The rules of the game world
  - The physics simulation
  - Knows nothing about user input or display
- View
  - Renders the model to the screen
  - Uses the model to know where to draw everything
- Controller
  - Handles user input and manipulates the model


Controller
- What did the player do?
- Time elapsed?
View
- Renderer and Sprites
Model
- Rules
- Objects

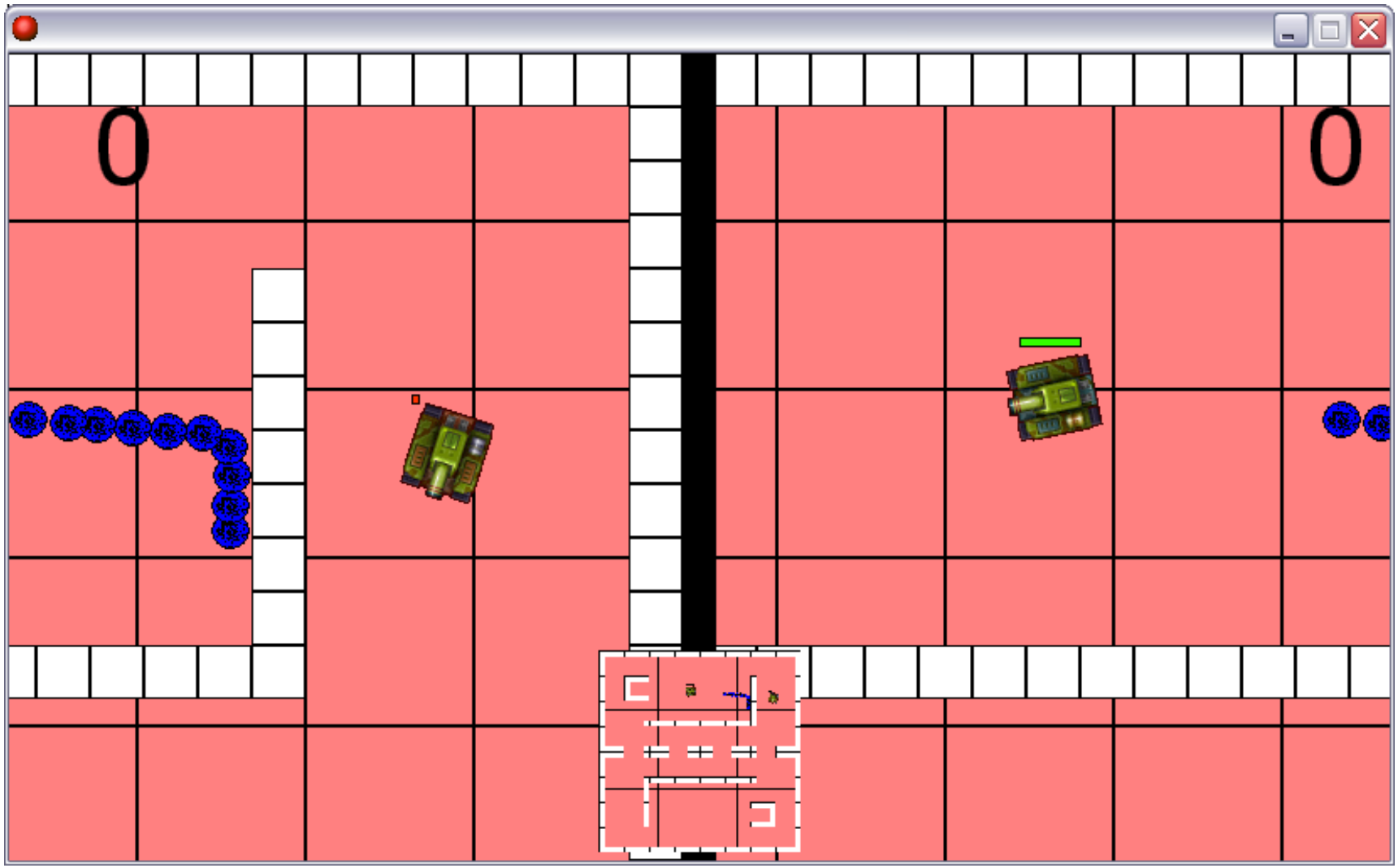
Quake MVC Architecture

- Model
  - An abstract 3d environment
  - Positions and orientations change over time
- View
  - Render the 3d environment
  - Display complex avatars and animations
  - Fancy effects
- Controller
  - Tell the model that I want to move, shoot, jump
  - Tell the model that 1/50th of a second has elapsed


Player
-(x,y,z)
-velocity
-Klesk avatar
-Railgun
-No ammo
-Bounding box

Why MVC is popular / good

- Nice modular design
  - Decouple game design from renderer
- Game world logic is bundled in the model
- Changes to the renderer / graphics do not affect the rest of the game
- Easily supports different input controllers and/or bots and AI
- Helpful when we think about networked games


0
0

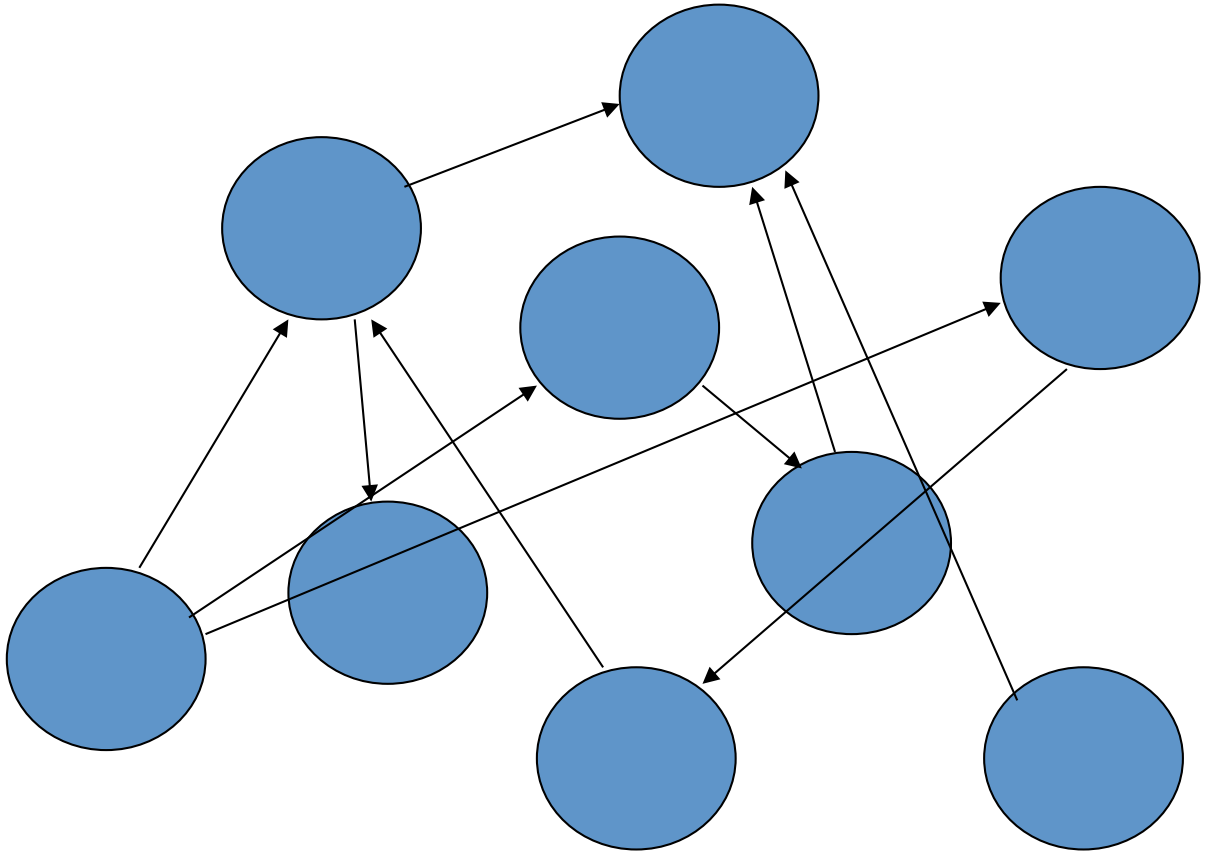

Gamer's Hell

Structuring the Model

- Model
  - Objects
  - Rules
  - Together create game world and state
- Objects need to communicate with one another
- Objects need to obey rules and procedures
- Objects need to be able to do things by themselves
- How do we structure this sensibly?

Direct Communication

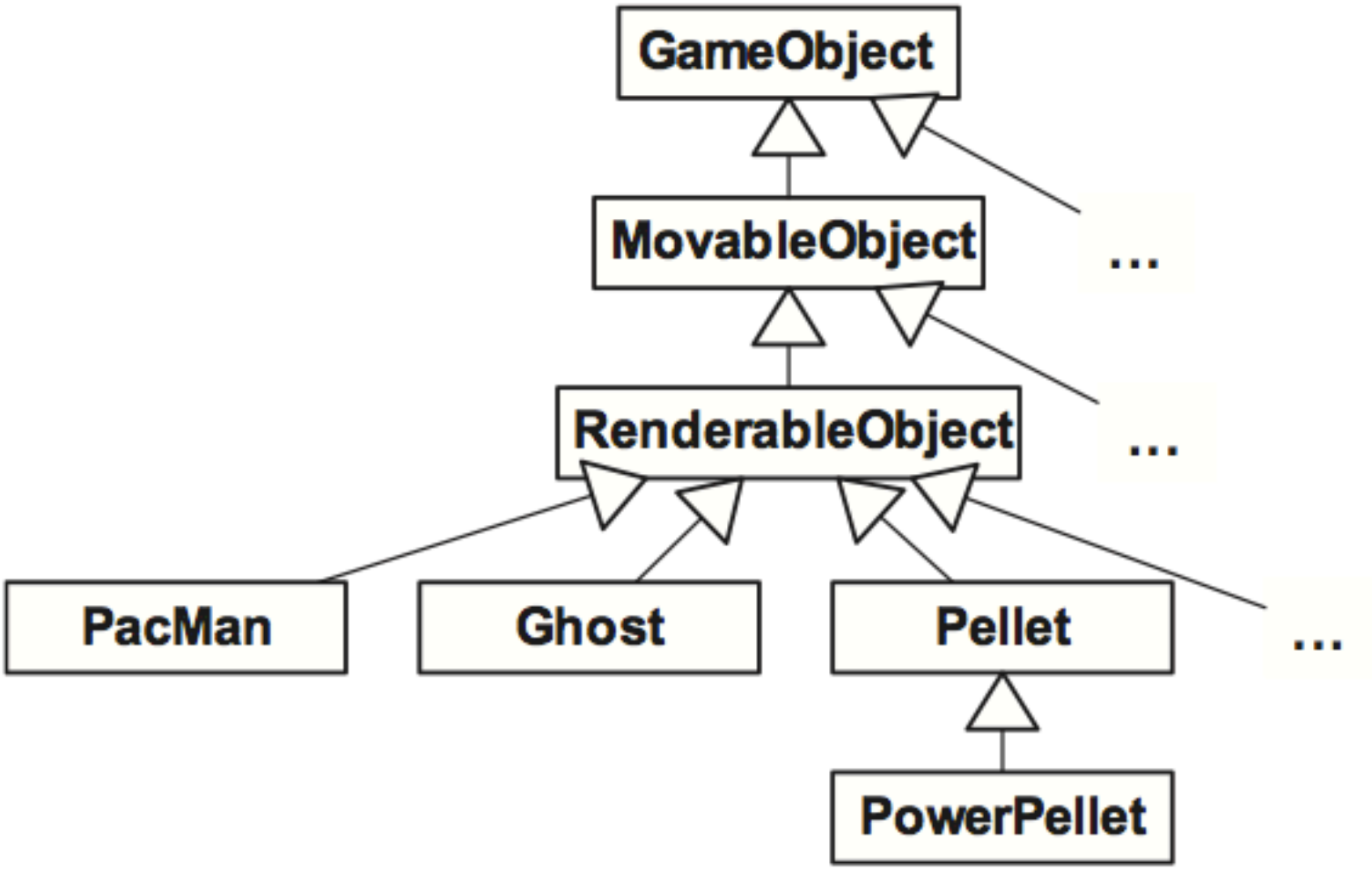
- Object A attempts to pick up object B
  - Who checks if B can be picked up
  - Which functions A must call in B to reflect pick-up
  - Many conditional statements
- Bullet hits player
  - Who decides what happens?
  - The player is damaged
  - The bullet is destroyed
  - Where do we put the logic?
- Why is this a poor design choice?
  - Exponential complexity
  - Every object needs to know how to interact with every other object
  - Very time consuming to add new objects

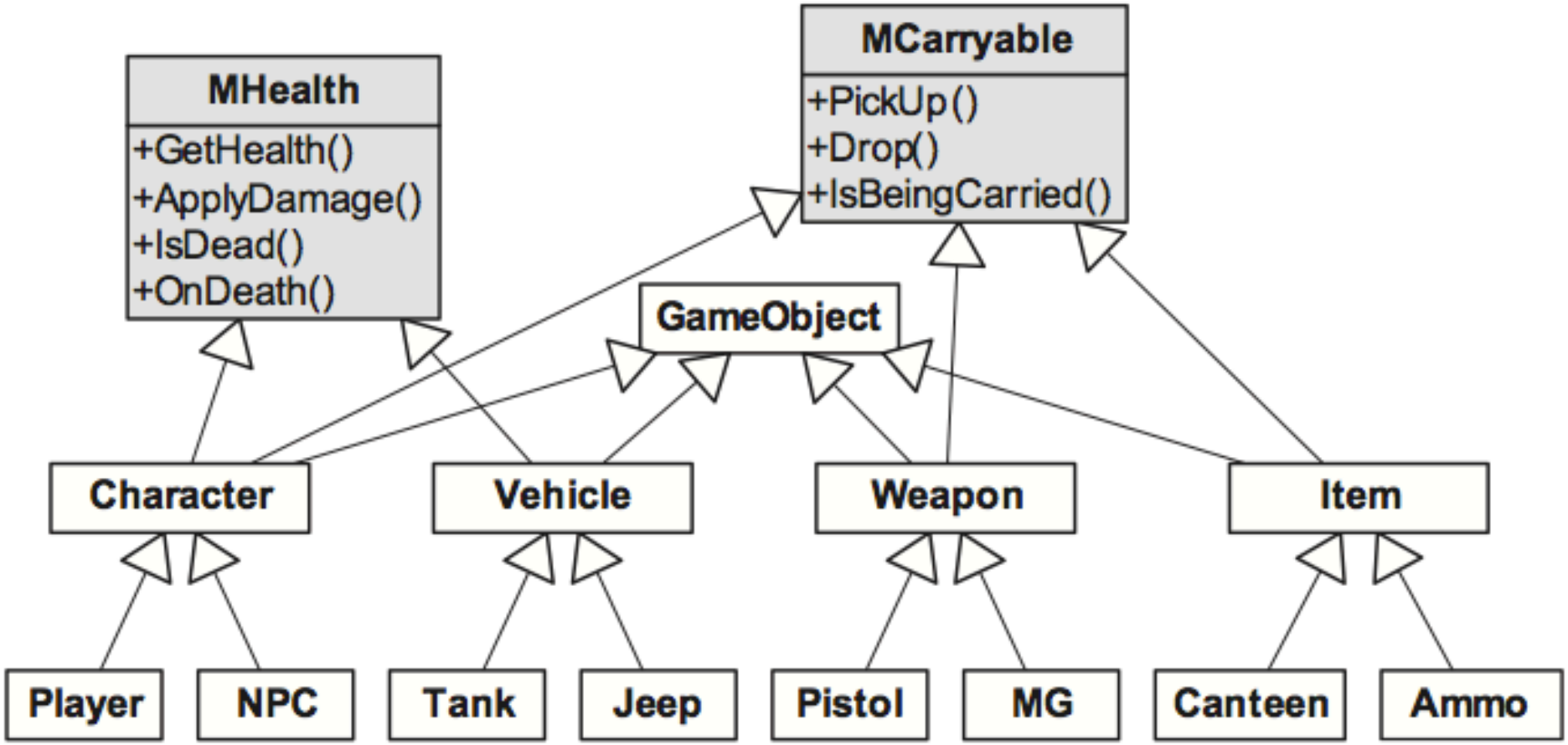


Encapsulation and Inheritance

- Generalise the kinds of functions that objects must respond to, and encapsulate this functionality within the object
  - Build an object hierarchy
  - Objects look after themselves – adding new objects is trivial
- Functions
  - Update() - Calculate where I am now
  - Render() - How I am drawn on screen
- Subsystems **iterate** object collections, but are dumb!

```
for(Object o : all the objects)
{
    o.update();
    o.render();
}
```


GameObject
MovableObject
...
RenderableObject
...
PacMan
Ghost
Pellet
...
PowerPellet

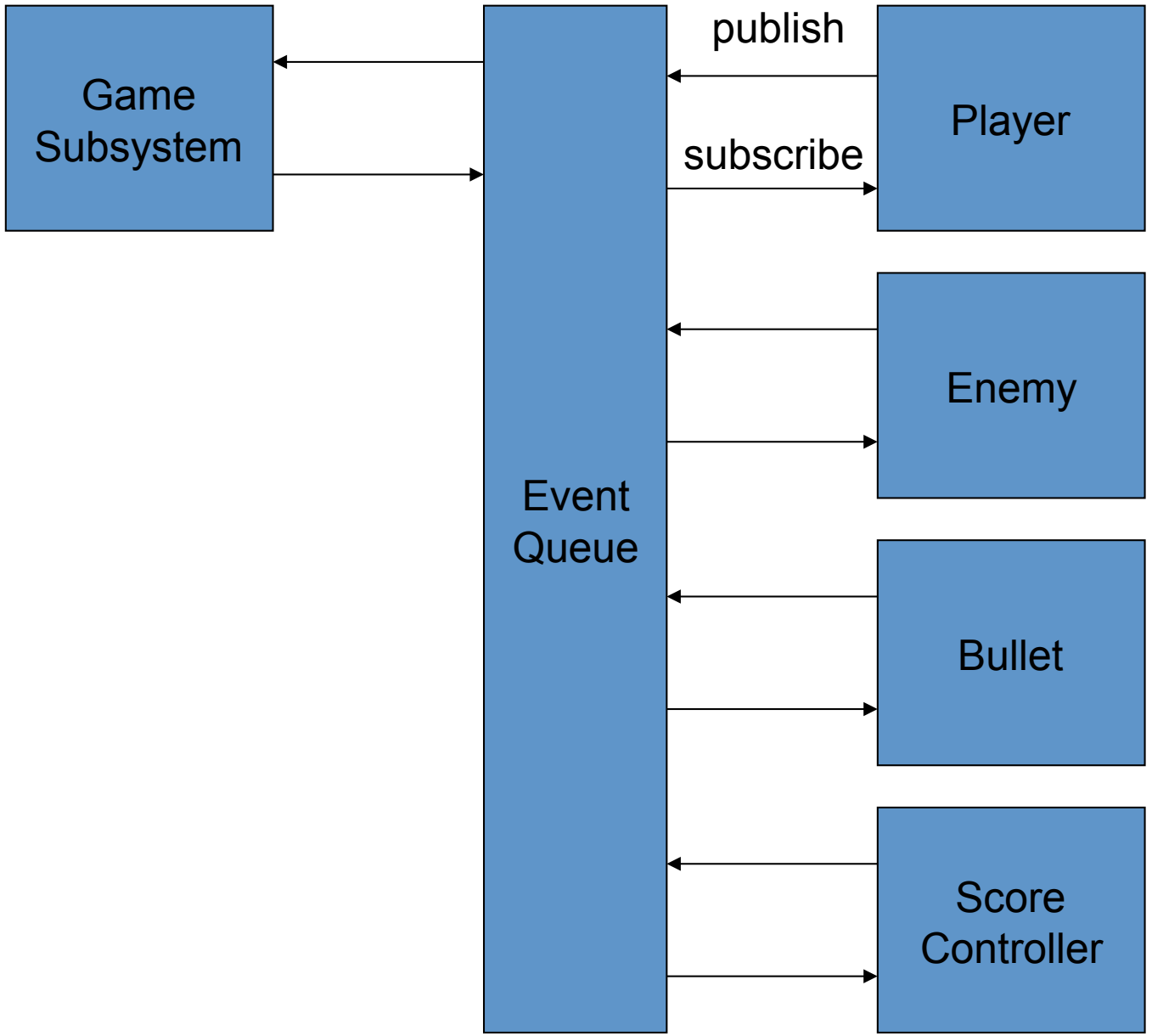

MCarryable
+PickUp()
+Drop()
+IsBeingCarried()
MHealth
+GetHealth()
+ApplyDamage()
+IsDead()
+OnDeath()
GameObject
Character
Vehicle
Weapon
Item
Player
NPC
Tank
Jeep
Pistol
MG
Canteen
Ammo

Events

- Games are inherently **event** driven
  - Anything of interest that happens
    - Explosion goes off, coin being picked up
  - Need a mechanism to…
    - Inform relevant objects that something has happened
    - Respond (**handle**) the event in some way
- "Event subsystem"
  - Often provided by a game engine to make our lives easier

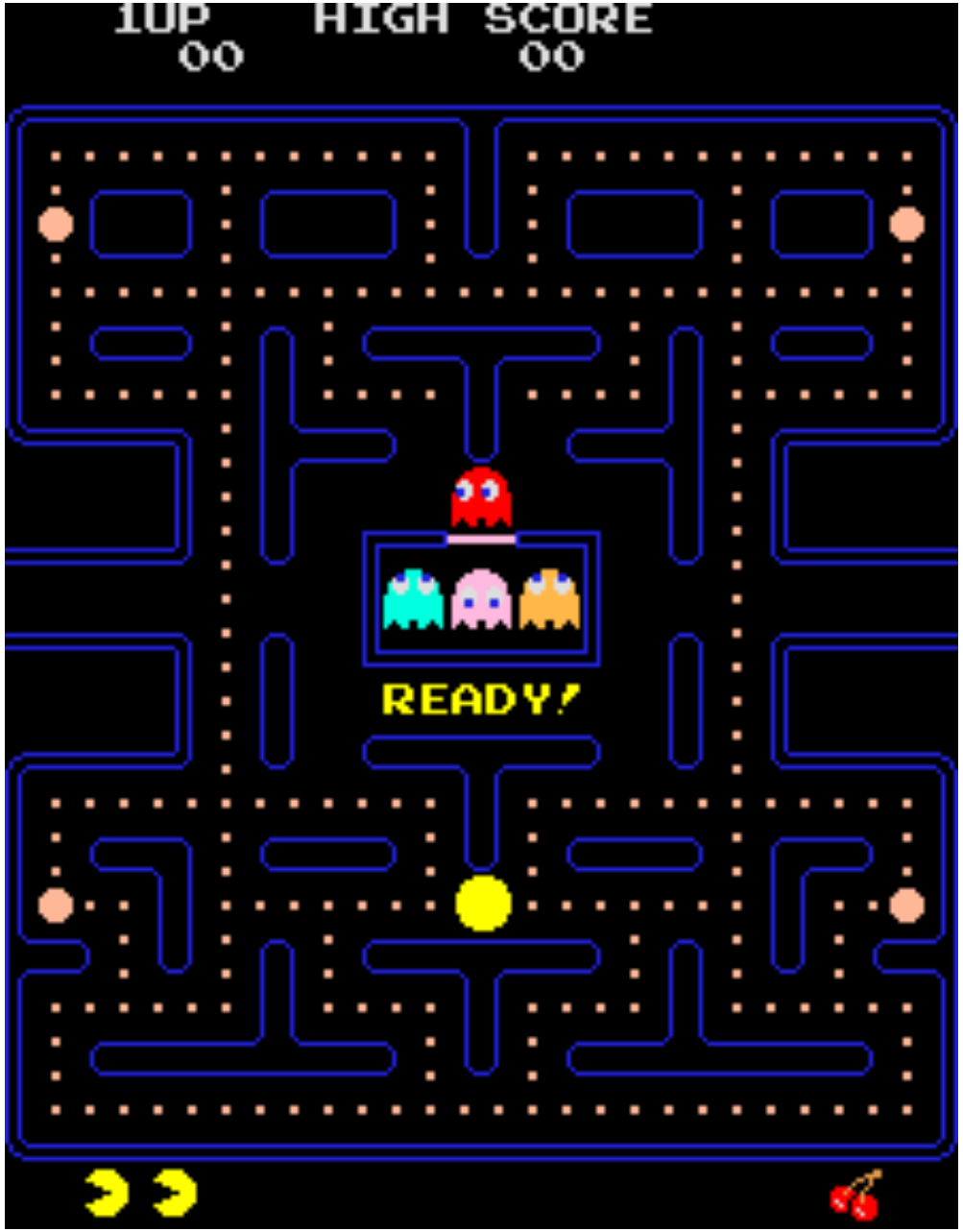
Events

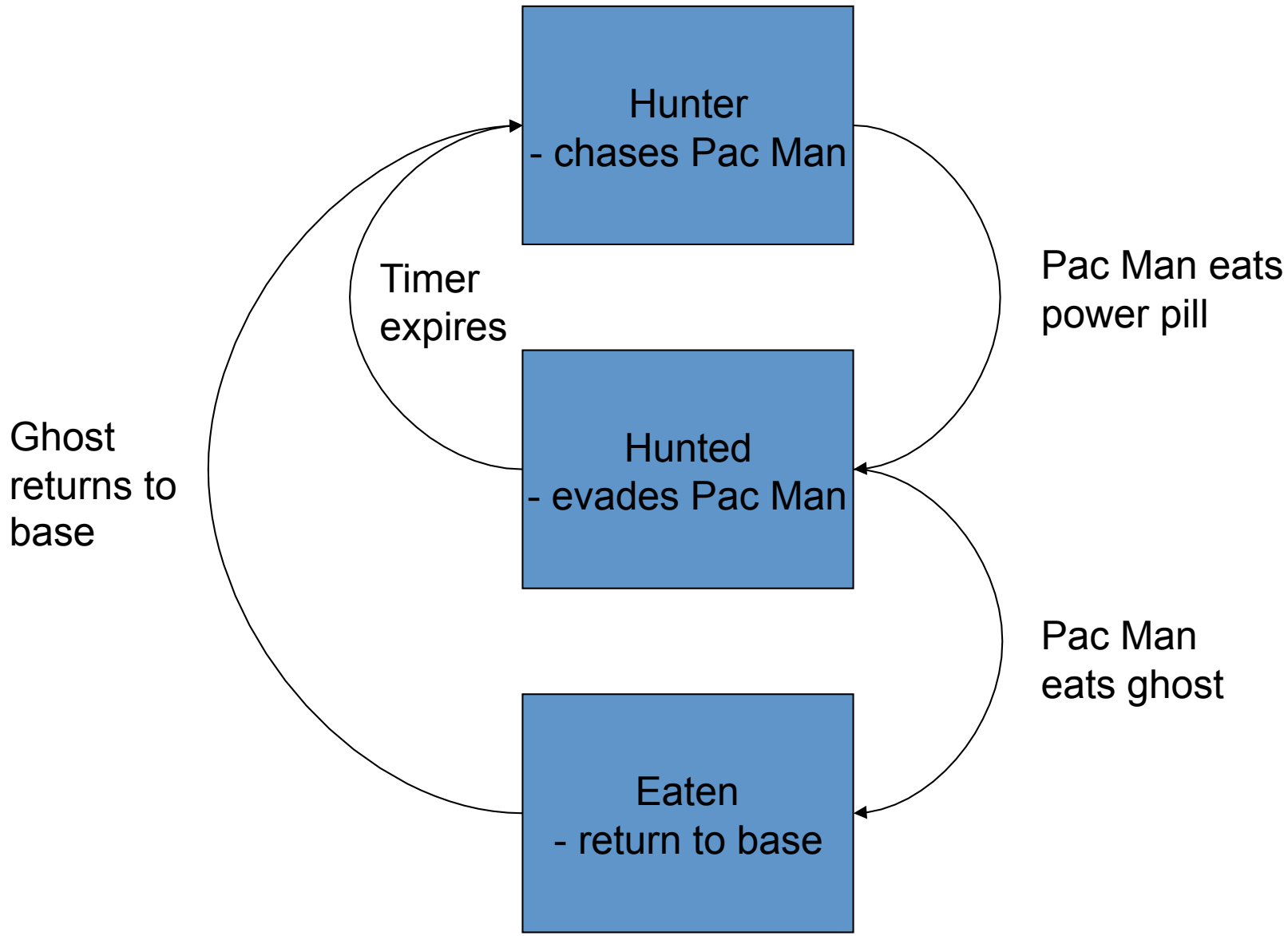
- Physics simulation
  - A bullet has collided with the player
  - Could tell all interested objects directly?
    - Bullet, player, scoreboard, health, walls, floor, sky…
- Broadcast an **event**
  - Objects **listen** for particular events to **handle**
  - Ignore events we don't care about
    - Scenery doesn't care about the score
  - Makes it easy to add new events
  - Objects can prioritise events in different ways
  - Dynamically register and unregister interest in events
    - Changing context


Game Subsystem
Event Queue
publish
subscribe
Player
Enemy
Bullet
Score Controller

Finite State Machines

- Events allow objects and subsystems to talk to one another
- FSMs allow objects to be autonomous
  - Evolve from “exploding” to “destroyed”
  - Artificial Intelligence / “bots”
- Literally
  - A **finite** number of **states**
  - **Actions** allow the object to **transition** between **states**


1UP
00
HIGH SCORE
00
READY!


Hunter
- chases Pac Man
Timer
expires
Pac Man eats
power pill
Hunted
- evades Pac Man
Ghost
returns to
base
Pac Man
eats ghost
Eaten
- return to base

FSM code

```
update()
{
    switch(state)
    {
      case hunting:
        chasePacMan();
      case hunted:
        evadePacMan();
       case eaten:
        returnToBase();
    }
}
```

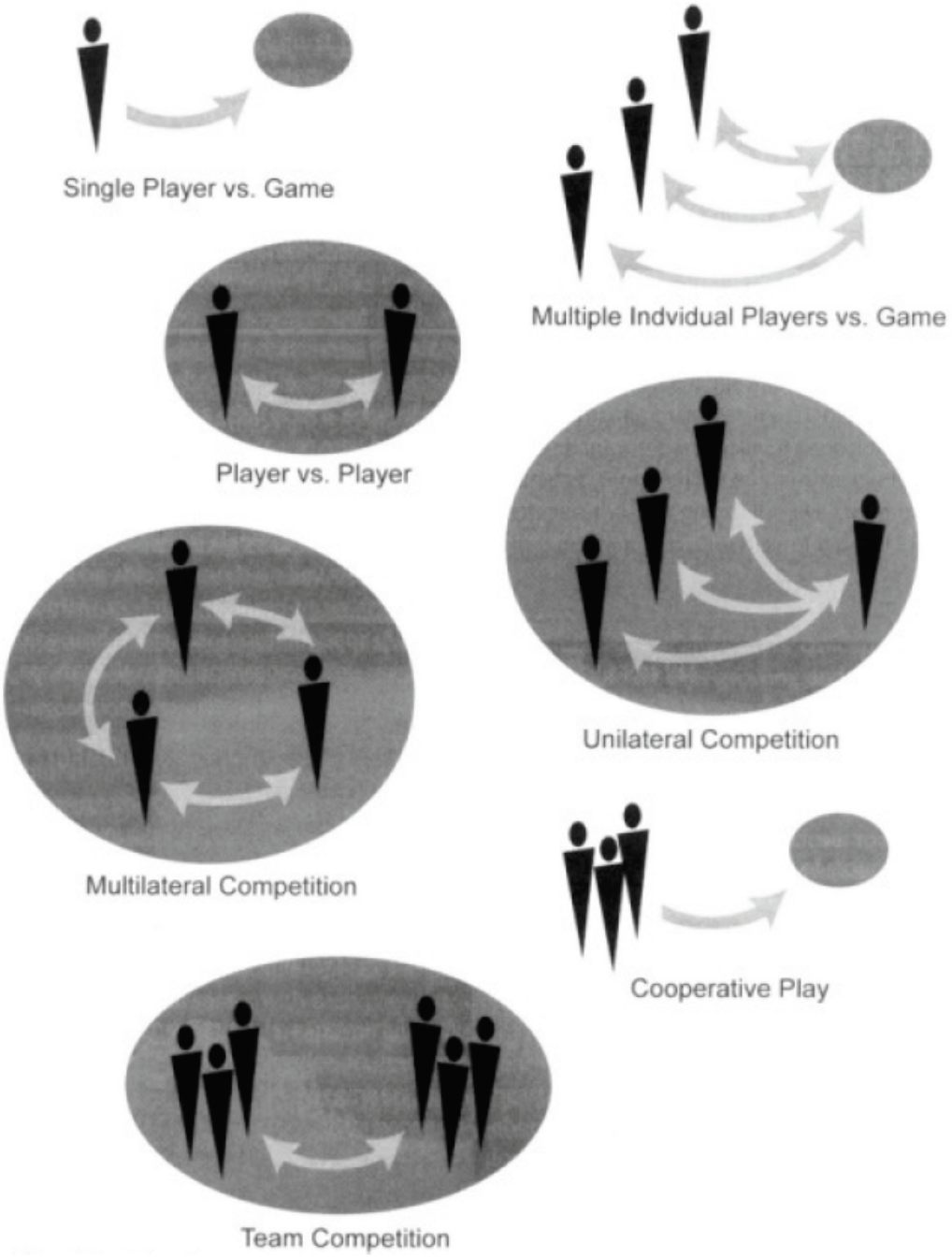
```
doEvents()
{
    event = getNextMessage();

    switch(event)
    {
      case eatenPill:
        setState(hunted);
      case reachedBase:
        setState(hunting);

      …
    }
}
```

References

- Game Engine Architecture (Jason Gregory, 2009)
- Quake source code
  - https://github.com/id-Software/Quake-III-Arena
  - C code, but can you spot the structure?

Multiplayer Games

- Single Player
  - Pre-defined challenges
  - Artificial Intelligence controlled opponents using Finite State Machines
  - Simple MVC design
- Multi-Player
  - More than one player can play in the same environment at the same time
  - Interaction with other players forms a key challenge of the game play
  - How can we build such a system?
  - What are some of the issues that arise?


Single Player vs. Game
Multiple Indvidual Players vs. Game
Player vs. Player
Unilateral Competition
Multilateral Competition
Cooperative Play
Team Competition

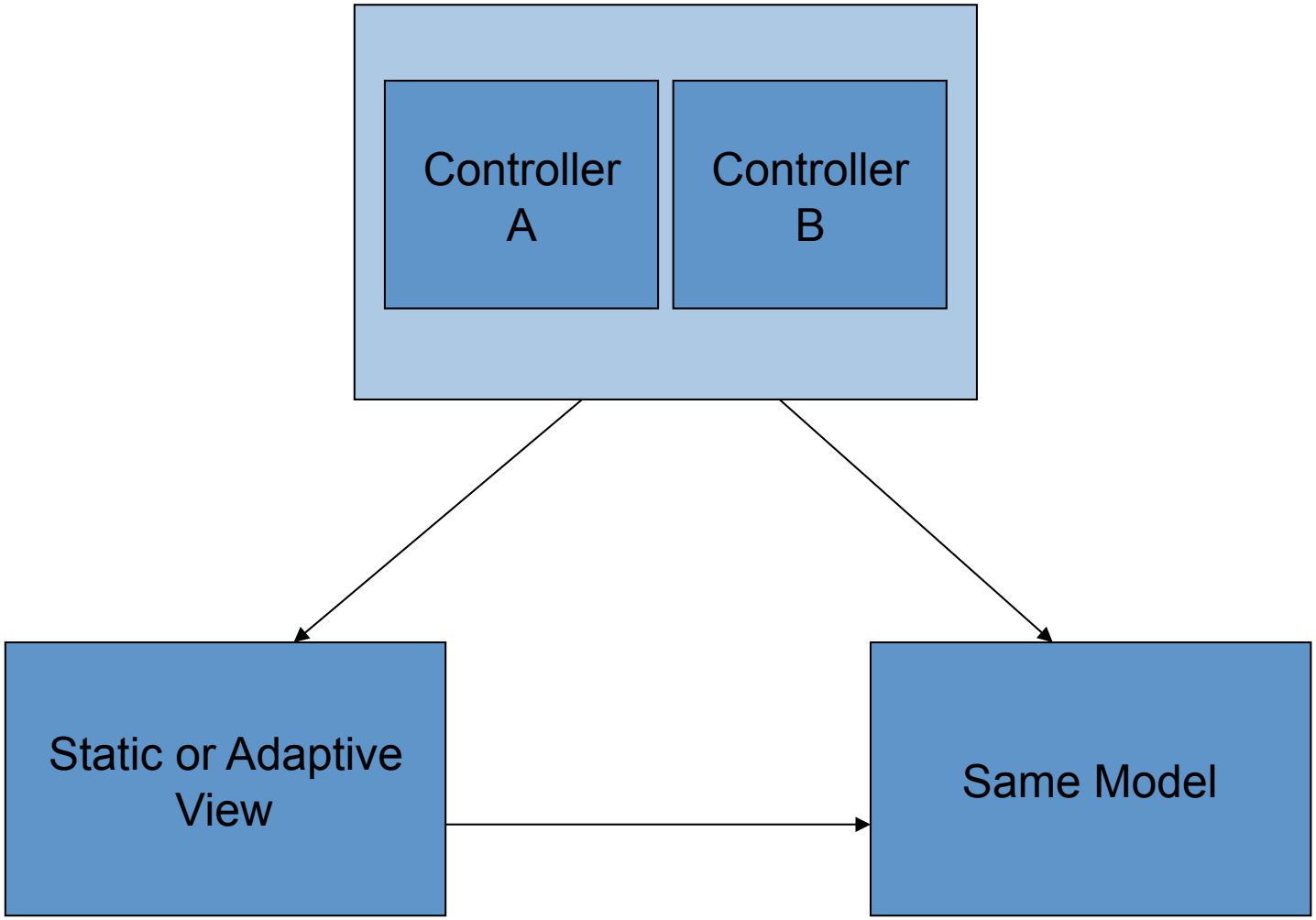
Where is the view?

- Local
  - Players are co-located
  - Share the same console / screen / pc
  - Share or split screen into two or four sections
  - Arcade games, racing, fighting, co-operative shooters
- Networked / Online
  - Players are physically separated
  - Game play is shared over the network / Internet
  - Many combinations of players 2 -> ??
  - FPSs, MMORPGs

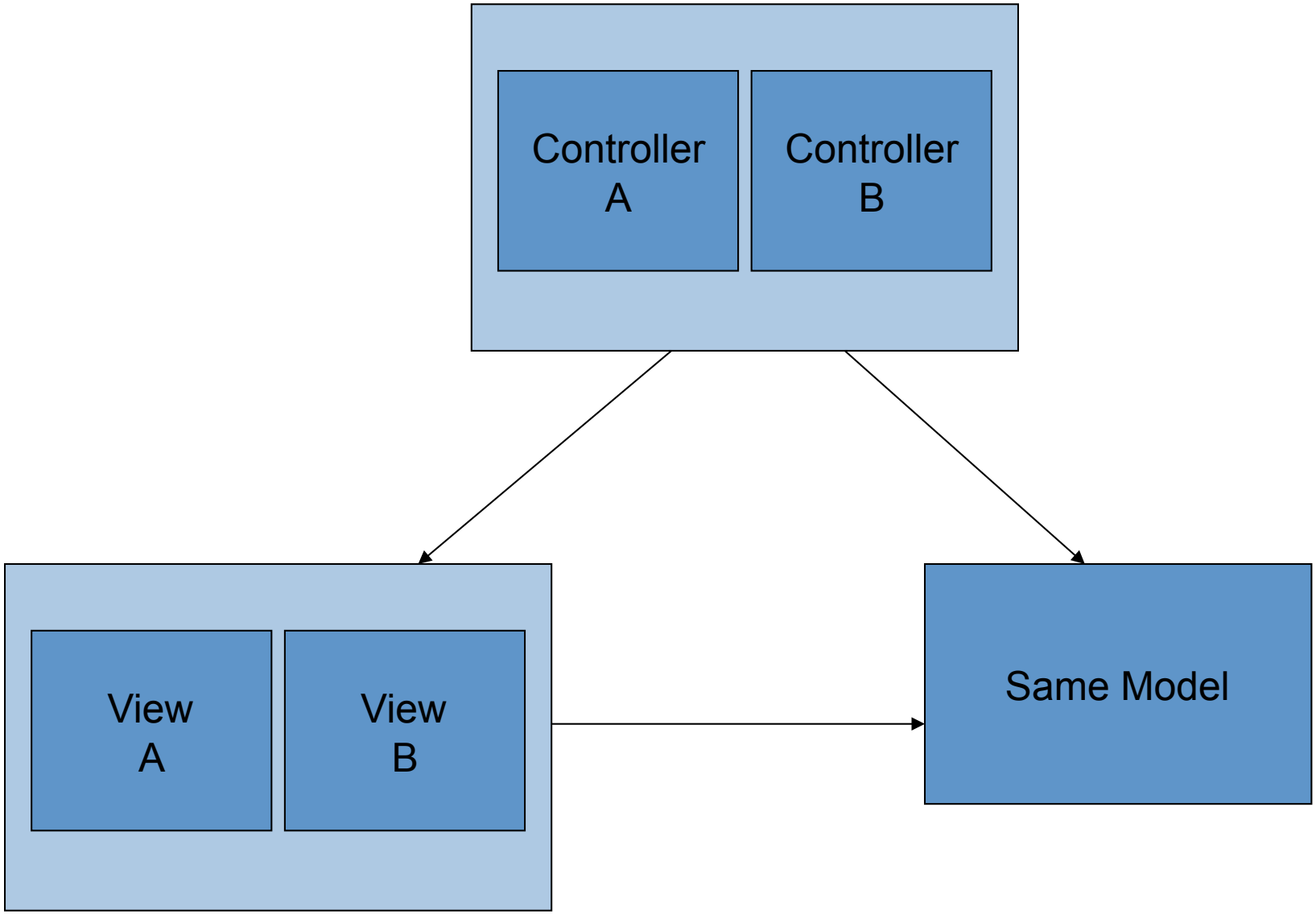
Static Shared View – Bomber Man

Adaptive Shared View – Street Fighter

Adaptive Shared View – Street Fighter


Controller A
Controller B
Static or Adaptive View
Same Model

Split-Screen View – Mario Kart


Controller
A
Controller
B
View
A
View
B
Same Model

Networked / Online Game Play

- Players are physically separate
- Where is the game?
- **Master** and **slave**
  - Usually two players, local network
- **Dedicated server** and **Clients**
  - Multiple players, local network, internet
- **(Peer-to-peer)**
  - Largely theoretical