# Game Dev: Entity Systems

Ricard Pillosu - UPC

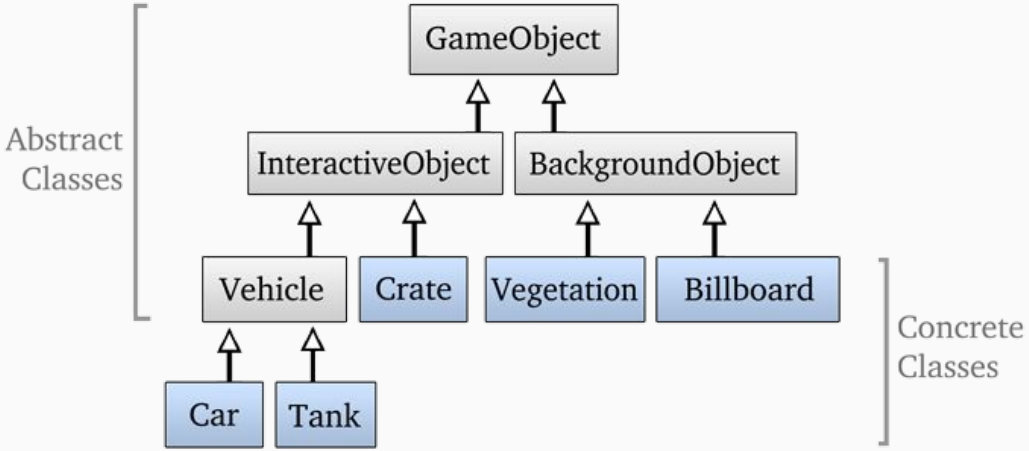
# Remember Enemy class from Project 1 ?

```h
class Enemy
{
protected:
    Animation* animation;
    Collider* collider;

public:
    iPoint position;

public:
    Enemy(int x, int y);
    virtual ~Enemy();

    const Collider* GetCollider() const;

    virtual void Move() {};
    virtual void Draw(SDL_Texture* sprites);
};
```

```cpp
Enemy::Enemy(int x, int y) : position(x, y), collider(nullptr)
{}

Enemy::~Enemy()
{
    if(collider != nullptr)
        App->collision->EraseCollider(collider);
}

const Collider* Enemy::GetCollider() const
{
    return collider;
}

void Enemy::Draw(SDL_Texture* sprites)
{
    if(collider != nullptr)
        collider->SetPos(position.x, position.y);

    App->render->Blit(sprites, position.x, position.y, &(animation->GetCurrentFrame()));
}
```

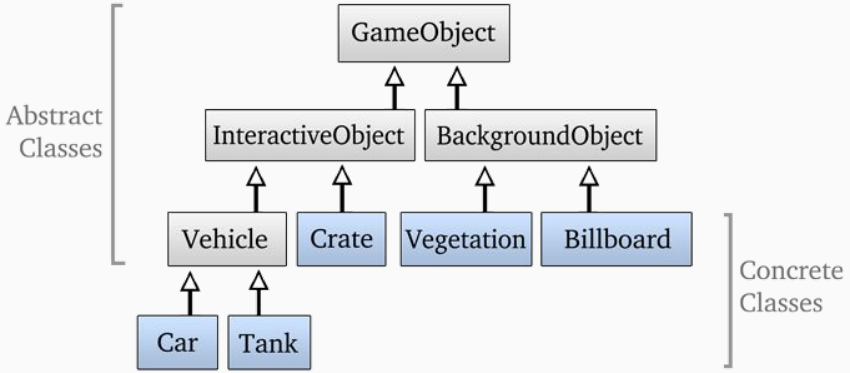
# Entity Systems

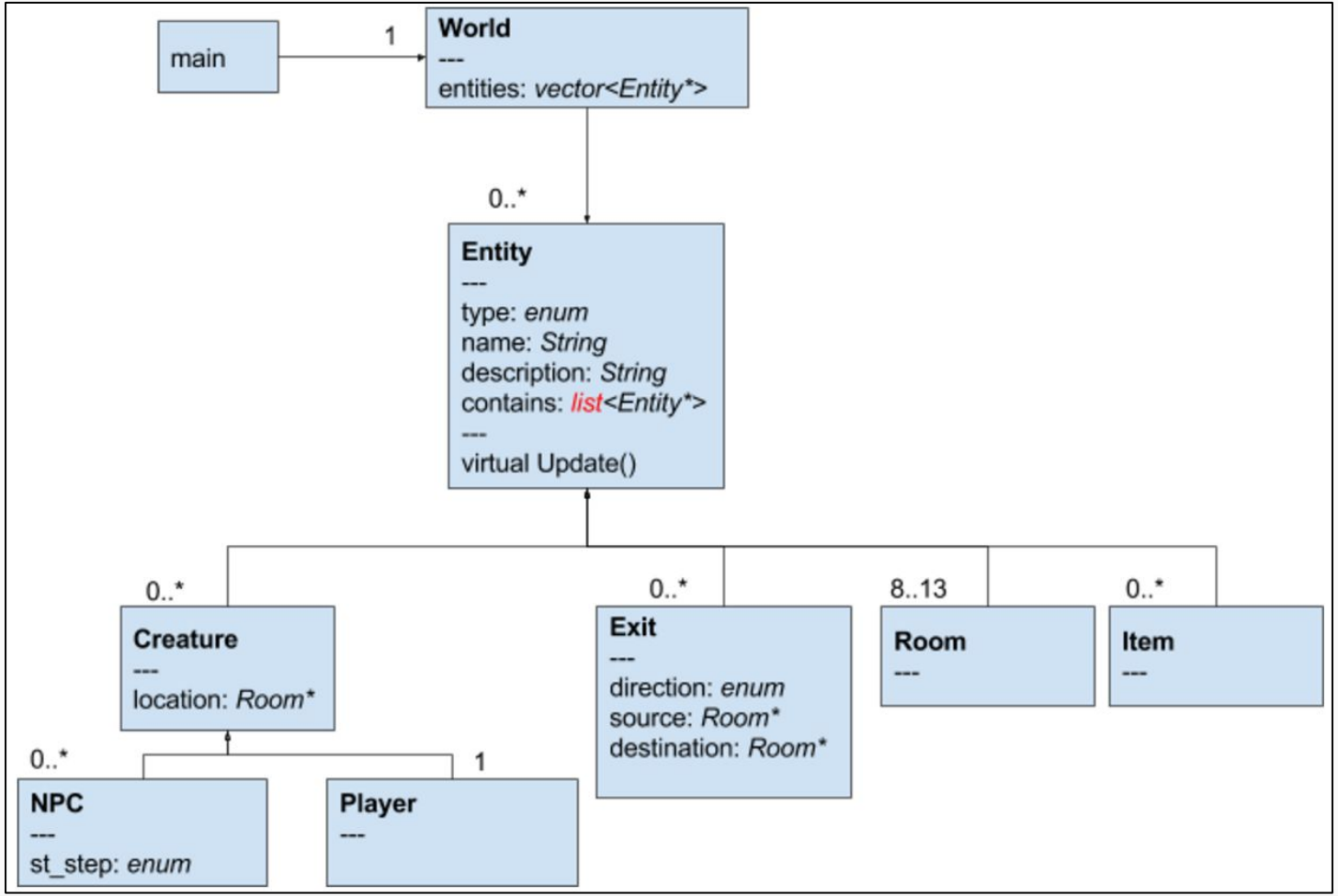
- Using OOP model we can describe all Entities in the game
- Exercise: Expand this structure to full UML with methods and properties:



# Entity Systems

- The main advantage is that we can distribute data and functionality
- Data as class properties and functionality as class methods:
  - GameObject to contain a position
  - Interactive to have move() method
  - Vehicle to have speed and turn angle
  - Tank to *have fire* cannon methods ?
  - Car to have a radio ?
  - Billboard to have a OrientToCamera()






Classical Zork Entity System (code here)

# Implementation: Timed Updates

```
bool EntityManager::Update(float dt)
{
	accumulated_time += dt;
	if(accumulated_time >= update_ms_cycle)
		do_logic = true;

	UpdateAll(dt, do_logic);

	if(do_logic == true) {
		accumulated_time = 0.0f;
		do_logic = false;
	}
	return true;
}
```

# Implementation: Entity Factory

```
enum class Types
{
	npc,
	player,
	room,
	exit,
	item,
	unknown
};
```

```
Entity* EntityManager::CreateEntity(Entity::Types type)
{
	static_assert(Entity::Types::unknown == 5, "code needs update");
	Entity* ret = nullptr;
	switch (type) {
		case Entity::Types::npc:    ret = new NPC();     break;
		case Entity::Types::player: ret = new Player();  break;
	}

	if (ret != nullptr)
		entities.push_back(ret);

	return ret;
}
```

# Implementation: Creating Entities

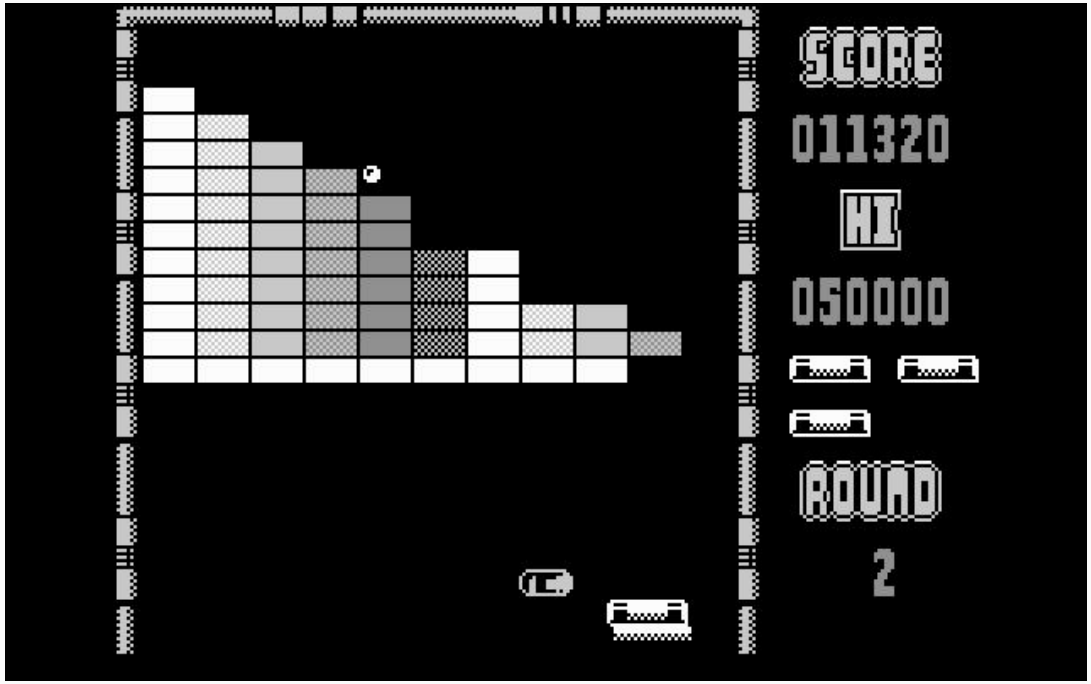
```
Entity::Entity(Types type) : type(type)
{}
```

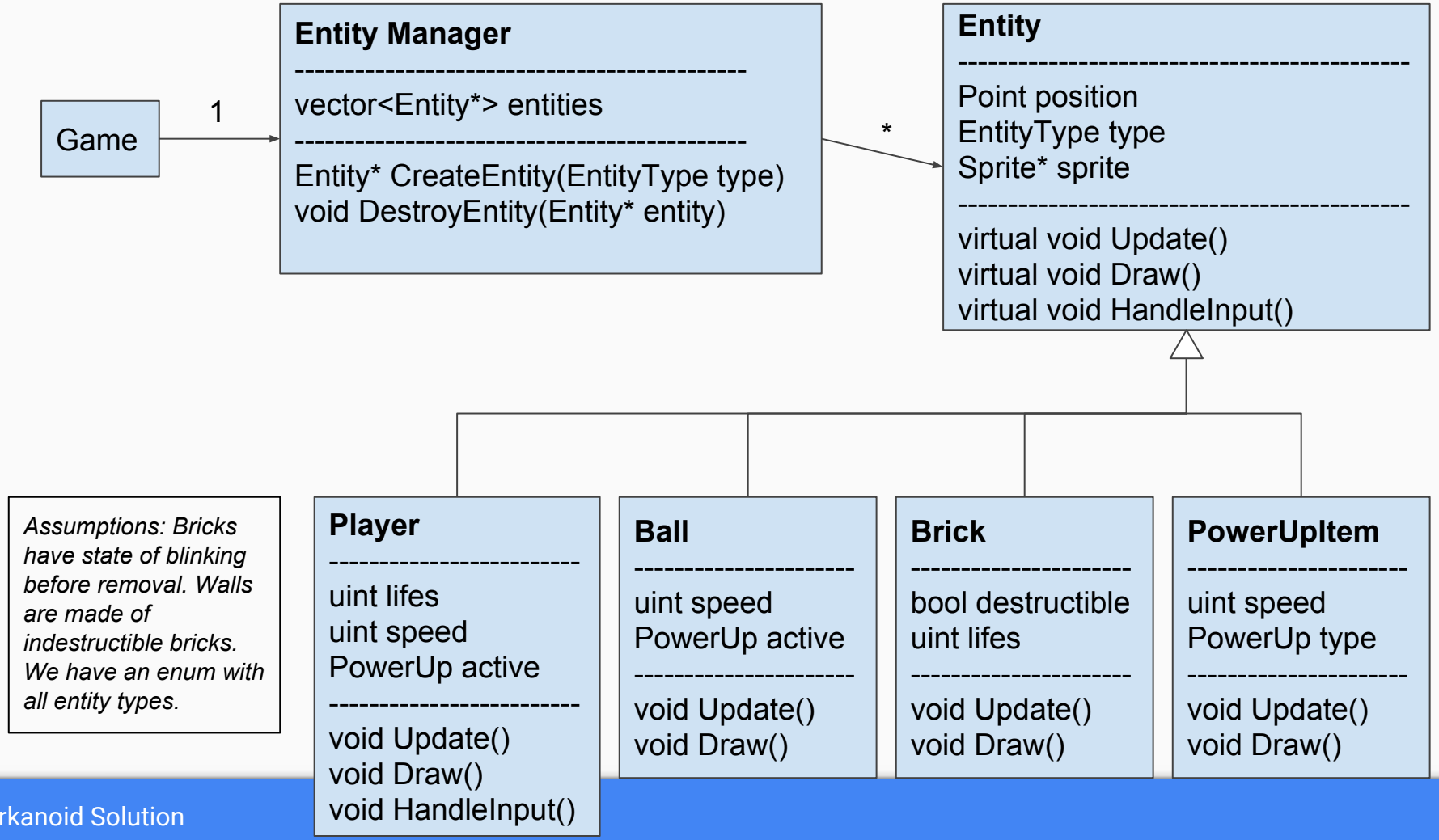
```
class Player : public Entity {
public:
	Player ();
	~Player ();
...
}
```

```
Player::Player() : Entity(Types::player)
{}
```

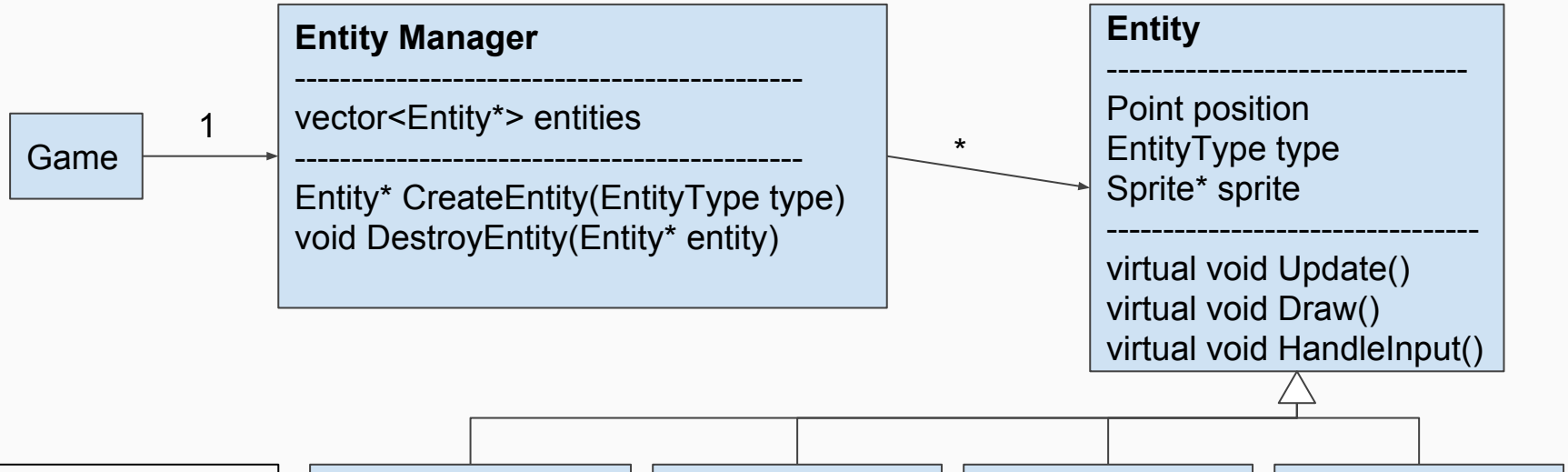
```
Player* player = (Player*) App->entities->CreateEntity(Entity::Types::player);
```

# Write the UML for Arkanoid Entity System




Game
1
Entity Manager
vector<Entity*> entities
Entity* CreateEntity(EntityType type)
void DestroyEntity(Entity* entity)
*
Entity
Point position
EntityType type
Sprite* sprite
virtual void Update()
virtual void Draw()
virtual void HandleInput()
Assumptions: Bricks have state of blinking before removal. Walls are made of indestructible bricks. We have an enum with all entity types.
Player
uint lifes
uint speed
PowerUp active
void Update()
void Draw()
void HandleInput()
Ball
uint speed
PowerUp active
void Update()
void Draw()
Brick
bool destructible
uint lifes
void Update()
void Draw()
PowerUpItem
uint speed
PowerUp type
void Update()
void Draw()

# Write the UML for Mario Entity System


Game
1
Entity Manager
vector<Entity*> entities
Entity* CreateEntity(EntityType type)
void DestroyEntity(Entity* entity)
*
Entity
Point position
EntityType type
Sprite* sprite
virtual void Update()
virtual void Draw()
virtual void HandleInput()
Assumptions:
Background is not an entity. We have an enum with all entity types.
Player
uint lifes, coins
uint speed
bool riding
PowerUp active
int animation
vec2 direction
void Update()
void Draw()
void HandleInput()
Mount
bool riding
int animation
bool mounth_full
void Update()
void Draw()
Enemy
int animation
int state
bool destructible
vec2 direction
int speed
bool eatable
void Update()
void Draw()
Item
PowerUp type
uint speed
uint animation
bool pickable
bool eatable
void Update()
void Draw()

# References

- Entity systems as explained here are considered old fashioned nowadays
- Component Based Systems are an evolution of Entity Systems (not for this subject, but interesting read anyway)
- More info here

# Homework

Write down the UML for elements in this screenshot.

Code a simple *Entity System* that represents those entities.