

<sup>1</sup> App

<sup>0,1</sup> Entity Manager

```
j1EntityManager();
virtual ~j1EntityManager();
bool Awake(pugi::xml_node&);
bool Start();
bool PreUpdate();
bool UpdateTick(float dt);
bool PostUpdate();
bool CleanUp();
bool CleanEnemies();

bool Load(pugi::xml_node&);
bool Save(pugi::xml_node&) const;

public:
Entity* CreateEntity(entity_type type, enemy_type enemy);
Enemy* CreateEnemy(enemy_type type);
void DestroyEntity(p2List_item<Entity*>* item);

private:
bool UpdateEntities(float dt, bool mustCheckLogic);
pugi::xml_node LoadConfig(pugi::xml_document&) const;

public:
p2List<Entity*> entities;
Player* player;

private:
ushort logicPerSecond;
float accumulatedTime;
bool mustCheckLogic = false;
```

<sup>0,\*</sup> Entity

<sup>0,1</sup> Player

```
struct movement_flags
bool movingUp;
bool movingRight;
bool movingLeft;
bool movingDown;

public:
p2SString name;
p2SString folder;
bool active;
bool mustDestroy = false;

protected:
fPoint position;
fPoint centerPosition;
movement_input input;
movement_flags movement;
p2SString textureName;
SDL_Texture* graphics = nullptr;
iPoint spriteSize;
Animation* animPtr = nullptr;
SDL_Rect animRect;
fPoint speed;
fPoint maxSpeed;
fPoint acceleration;
float gravity;
bool lookingRight;
Collider* hitbox;
SDL_Rect hitboxOffset;
int life;
uint maxLife;
fPoint hurtSpeed;
bool dead;

enum class entity_type
NONE = -1,
NPC,
PLAYER,
PLAYER_ATTACK,
ENEMY,
ENEMY_ATTACK,
ITEM,
CHECKPOINT,
MAX_TYPES

public:
entity_type type;

protected:
virtual void Awake(pugi::xml_node&);
virtual ~Entity();
virtual void Init();
virtual bool Awake(pugi::xml_node&);
virtual bool Start() { return true; }
virtual bool PreUpdate() { return true; }
virtual bool UpdateLogic(float dt) { return true; }
virtual bool UpdateTick(float dt) { return true; }
virtual bool Update() { return true; }
virtual bool PostUpdate() { return true; }
virtual bool PostUpdate() { return true; }
virtual bool CleanUp() { return true; }
virtual bool Load(pugi::xml_node&) { return true; }
virtual bool Save(pugi::xml_node&) const { return true; }

public:
virtual entity_type GetType() const;
virtual fPoint GetPosition() const;
virtual fPoint Entity::GetCenterPosition() const;
virtual bool InsideRadius(fPoint targetPosition, iPoint radius);
virtual fPoint GetSpeed() const;
virtual fPoint GetAcceleration() const;
virtual Collider* GetCollider();
virtual collision_type OnCollision(Collider*, Collider*) { return (collision_type)-1; }
virtual uint GetLife() const;
virtual void Kill();
virtual void Hurt();
virtual bool IsDead() const;

protected:
virtual void CheckInput() {};
virtual void Draw(SDL_Rect* animRect) const;
virtual void ImportSpriteData(const char* spriteName, sprite_data* sprite, pugi::xml_node&);
virtual void CheckMovement();
virtual void CheckState() {};
virtual void ApplyState() {};
virtual void Move(float dt) {};
virtual void UpdateHitbox() {};
virtual bool CheckOrientation(bool orientation);
virtual fPoint LimitSpeed();
SDL_Rect ReshapeCollider(sprite_data sprite);
```

```
enum class player_state
IDLE,
CROUCHING,
RUNNING,
JUMPING,
FALLING,
SLIDING,
HURT,
ATTACKING

struct attack_data
SDL_Rect offsetRight;
SDL_Rect offsetLeft;
ushort startAttackFrame;
ushort finishAttackFrame;

Player();
~Player();

collision_type WallCollision(Collider* c1, Collider* c2);

public:
bool CameraFree() const;
IsGod() const;
iPoint GetActivationRadius() const;
void ReturnToSpawn();

private:
void ImportAttackData(const char* spriteName, attack_data* attack, pugi::xml_node& first_sprite);
void ImportAllStates(pugi::xml_node&);
void ImportAllSprites(pugi::xml_node&);
void AllocAllAnimations();

Collider* CreateAttackCollider(attack_data attack);
SDL_Rect ReshapeAttackCollider(attack_data attack);

void Jump();
void Fall(float dt);
void LateralStop();
void Land();
void StandUp();
void Hurt();
void PlayerReset();
void DeathByPit();

void DebugInput();
player_state IdleMoveCheck();
player_state CrouchingMoveCheck();
...

void IdleEffects();
void CrouchingEffects();
...

fPoint GodModeMovement(float dt);
fPoint NormalMovement(float dt);

public:
bool debugMode;

private:
player_state status;
Collider* attackCollider;
SDL_Rect attackOffset;
bool wantAttack = false;
bool attackColliderCreated = false;
fPoint lastGroundPosition;
fPoint respawnPosition;
float normalAcceleration;
float slideAcceleration;
iPoint activationRadius;
bool airborne;
bool somersaultUsed;

uint attackTimer = 0;
ushort attackDelay;
uint deadTimer = 0;
ushort deathDelay;
bool fading = false;
float fadeDelay;

bool mustReset = false;
bool godMode;
bool freeCamera;

sprite_data idlesprite;
sprite_data runSprite;
...

uint runSfxTimer = 0;
int runSfxDelay;
bool playedSlideSfx;
bool playedHurtSfx;
```

<sup>0,\*</sup> Enemy

<sup>0,\*</sup> Slime

<sup>0,\*</sup> Bat

```
enum class enemy_type
NONE = -1,
BAT,
SLIME,
MAX_TYPES

enum class enemy_state
IDLE,
PATROLING,
FOLLOWING,
ATTACKING,
FALLING,
HURT

protected:
bool canFly;
enemy_state status;
iPoint spawnPosition;
bool airborne;
iPoint detectionRadius;
iPoint attackRange;
bool playerDetected = false;
bool playerInRange = false;
bool wantToAttack = false;
uint attackTimer = 0;
uint hurtTimer = 0;
uint deadTimer = 0;
ushort attackDelay; //Time to stay attacking
ushort hurtDelay; //Time to stay hurt
ushort deathDelay; //Time before despawn
bool mustReset = false;

private:
enemy_type enemyType;
```

```
Slime();
private:
void DashAttack();

private:
ushort origMaxSpeed;
ushort dashSpeed;

sprite_data idleSprite;
sprite_data moveSprite;
...
```

```
Bat();
private:
//Character sprites
sprite_data followSprite;
...
```