GMTB Version 2 Engine Documentation

Class Descriptions

* AI Manager – Class - Singleton
  + Controls all AI specific routines
  + Primarily informs AIs of the player’s current location
* AManager – Abstract Class
  + Allows additional subroutine managers to be created by the game file and be loaded into the core manager in the engine.
* AnimatingEntity – Abstract class
  + Inherits from Entity
  + Handles animation cycle
* Camera2D – Class
  + Overrides default camera to produce a player following camera
* Collidable – Interface
  + Gives Collision manager access to collision methods and variables in an entity
  + Implement this interface in an entity to reveal it to the Collision Manager
* CollisionManager – Class - Singleton
  + Primary Collision detector
* CoreManager – Class – Singleton
  + Runs all other managers required for gameplay
  + Suspends manager operation based on gamestate
* DialogueBox – Class – Singleton
  + Draws text on screen during dialogue
* Door – Class
  + Colidable object that triggers new level
* Entity – Abstract Class
  + Main class for physical objects
  + Includes methods for creation, basic collision and object disposal
* EntityManager – Class – Singleton
  + Main manager for creating and disposing entities
    - Note: Disposal not quite perfect
* FullArtRenderer - Class
  + Class to render a characters full screen artwork
    - Not finished
* GameOver – Class
  + Game over screen displayer
  + Runs as a menu
* Global – Static Class
  + Storage for global variables
* hasProximity – Interface
  + Gives Proximity Manager access to the proximity box of an object
    - Used to determine if the player is near an object – separate to collision
* HidingPlace – Class
  + Object the player can hide in
* IEntity – Interface
  + Main interface for Entity class
* IInventory – Interface
  + Interface for Inventory class
* IItem – Interface
  + Interface for Item class
* ILevel – Interface
  + Interface for Level classes
* Input – Class -Singleton
  + Main input detection and manager
* Inventory – Class
  + Records all collected items, gives them appropriate display location in inventory display.
* IWall – Interface
  + Interface for invisible walls
* Level – Abstract Class
  + Level descriptors inherit from this
* LevelManager – Class -Singleton
  + Lists all available levels during initialization
  + Handles level loading
* MainMenu – Class
  + Displays main Menu
  + Calls game start routine
* MenuManager – Class – Singleton
  + Ensures only one of each menu is created
* PauseMenu – Class
  + Same as Main
* Player – Class
  + Main player class
* ProximityManager – Class -Singleton
  + Same as collision manager but triggers proximity events
* RoomManager – Class – Singleton
  + Renders background
* SaveLoadManager – Class – Singleton
  + Controls file saving and loading
* SaveData – Struct – Serializable
  + Save file contents
* SceneManager – Class -Singleton
  + Controls creation of entity textures
  + Calls all entities to be drawn on command
* Script – Class – Singleton
  + Loads Dialogue from entity
  + Sends Dialogue line by line to DialogueBox
* SolidObject – Class
  + Unmoving object with collision
* AllAI – Class
  + Parent class for AI characters
  + Gives basic collision and Idle behaviour
  + Provides inheritance from the Animating Entity class
* FriendlyAI – Class
  + Nonviolent AI
  + Doesn’t move
  + Interacts with player
  + Triggers dialogue on action by player
* HighLevelAI – Class
  + Hostile AI with higher difficulty
  + Pursues player for long time
  + Large area of activation
* HostileAI – Class
  + Parent class for all hostile AI
  + Provides default pursuit and patrol behaviour
* IAI – Interface
  + Main interface for all AI
* INeutralAI – Interface
  + Alternate interface for neutral AI
* JumpScare – Class
  + AI with single behaviour to sprint at player on first level load
  + Doesn’t react on subsequent level loads
* LowLevelAI – Class
  + Hostile AI with low difficulty
  + Pursues player for short duration
  + Small area of activation
* NeutralAI – Class
  + Unmoving AI
  + Changes facing direction to look at player’s current location
  + Easter egg – Will talk if prodded by player enough times

Class Interactions

Player Creation process

EntitiyManager

IEntity newEntity<T>(PlayerIndex pPlayerNum) where T : IEntity, new()

{ Create PlayerObject

Give PlayerObject UID

Add PlayerObject to master List

Return PlayerObject

}

GameManager

InitializeGame()

{ Call Entity Manager to create PlayerObject

Pass PlayerObject and coordinates to SceneManager

Call LevelManager to start L1

}

LevelManager

NewLevel(string LevelID)

{ Find LevelID and Begin }

SceneManager

newEntity(IEntity createdEntity, int x, int y)

{

Add Entity to Master List

Load Entity Texture from itself

Position at X,Y

}

Player

Constructor ()

{ Call Parent constructor

Do own thing }

Texture { Get {return mTexture}}

Entity

constructor ()

{ Do own thing }

AnimatingEntity

constructor ()

{ Call Parent constructor

Do own thing }

Details from second year presentation

Entity Management

* All our Entities are created using the EntityManager object, the level class simply calls the EM’s create method when it wants to make a new Entity.
* Each Entity is given a Unique ID by the EM which the level manager is informed of.
* When a level is unloaded, the level manager calls the EM’s remove method and tells it the UID of the Entity to be removed.

Scene Management

* All our Entities are placed in the scene by the Scene Manager.
* They are then Updated and Drawn by the SM.
* The Scene Manager keeps a SceneGraph which holds all the currently visible Entities and an AllEntities List which holds every created Entity that may come back on the screen.
* It’s draw method also changes based on the current GameState within the Kernel, EG the Dialogue Box only needs to be drawn when in the Dialogue State.

Entity/Simulation Lifecycle

* Upon loading, the Kernel starts a Core Manager to run all the other managers from a single place.
* The Core manager can start and stop the other managers based on the GameState.
* The simulation starts with a Main menu, when the player clicks Play, it signals the Level Manager to load the first level.
* Each level descriptor class has an Initialise method that stores all the Entities for that level, it calls the Entity and Scene managers to place them.
* Colliding with a door triggers the Level manager again but tells it to load the Level specified by the door.
* When a level is left, all the entities in that level are moved off screen and disabled.
* If the player reenters a level, the level manager checks to see if the entities are still loaded, if they are then they are moved back to position and reactivated, if they have been removed from memory, then they are simply recreated.

Collision Management

* There are two types of collisions in our engine, Collisions and Proximity.
* Each type has its own manager and Entities implement an additional Interface for each type of collision and subscribe to the Manager they need.
* When a collision is detected the manager runs either the Collision or Proximity method in the collided Entity through the respective Interface.

Behaviour Management

* Each type of AI is responsible for its behaviour.
* Their texture, position and default behaviour are set by the level class that loads them.
* Hostile AI have a shared behaviour of following, patrolling and searching but there are two subtypes which alter the Texture and the time they should search or follow for.

Input Management

* All Input is handled by the Input Manager.
* Objects that want to know of Input subscribe to the managers events.
* There are different Events for different types of Input, for example, the Kernel only subscribes to the Escape key event while the player subscribes to WASD event.

Improvements and additions from Version 2

* Additional Managers can be written in the game solution and inherit from an abstract class in the engine solution and automatically imported to the Core Manager.
* Partial XInput controller support, Xbox 360/Xbox One/Dual Shock 4 using DS4Windows driver. Still lacking analogue stick control.
* 2D player following camera can be implemented to override the default fixed camera.
* 8 Direction fully implemented over old “4 direction, 8 if buttons held in right order”.