Anthony Chen | Detian Shi | Nicholas Beaumont
Danielle Lee | Leina Sha | Natalie Diebold

ARCHITECTURE SPECIFICATION

Class Responsibility Collaboration Tables

COMPONENTS

PhysicsPlugin (abstract)

Description: The PhysicsPlugin class contains physics-related information, and interacts with other PhysicsPlugins. This class serves as the base class for every other PhysicsPlugin.

Justification: All Physics-based objects have a base set of methods and values that dictate their actions during collisions with each other and with pings, so they are defined here to be overridden later.

Classification: Component

Responsibility	Collaboration
Encapsulate physics parameters	Box2D, Farseer
Receive collision callback	GameEntity

ControllerPlugin (abstract)

Description: The ControllerPlugin class contains a framework that allows control over a GameEntity. This class serves as the base class for every other ControllerPlugin

Justification: All ControllerPlugins supply an action to the GameEntity, so this class defines available methods to be overwritten.

Classification: Component

Responsibility	Collaboration
Perform relevant part of game update loop	GameEntity
Destruct game entity during disposal	N/A

ControllerPlugin::PingCP

Description: The PingCP class contains the calculations for an expanding ping.

Justification: The ping entity behaves uniquely to every other object in game.

Classification: Component

Responsibility	Collaboration
Expand physics object	PingPP
Destroy game entity at end of lifetime	GameEntity

ControllerPlugin::PlayerCP

Description: The PlayerCP subclass contains necessary code for controlling the player through keyboard and gamepad interfaces.

Justification: Players must be able to control the hero

Classification: Component

Responsibility	Collaboration
Receive player input, update physics	GameEntity, PhysicsPlugin
Spawn player ping	GameEntity, PingCP, PingGP, PingPP

ControllerPlugin::AICP

Description: The AIPlugin subclass contains all necessary code for processing of the AI. Differentiation in behavior is achieved through subclasses for every distinct AI.

Justification: All NPCs will follow this same AI, so it makes sense to include it as a plugin that can be used by all NPCs.

Classification: Component

Responsibility	Collaboration
Updates memory of targets	TargetData
Spawn AI ping	GameEntity, PingCP, PingGP, PingPP

AIControllerPlugin::HunterCP

Description: The HunterCP subclass contains the behavior logic for the hunter, including target selection, the Hunter's finite state machine and control code.

Justification: Hunter behavior and control is unique to Hunters, but they can use the AICP framework that controls target acquisition and other common tasks.

Classification: Component

Responsibility	Collaboration
Determine next state	HunterState
Move entity	HunterPP
Select current target	N/A

AIControllerPlugin::CritterCP

Description: The CritterCP subclass contains the behavior logic for the critter, including target selection, the Critter's finite state machine and control code.

Justification: Critter behavior and control is unique to Critters, but they can use the AICP framework that controls target acquisition and other common tasks.

Classification: Component

Responsibility	Collaboration
Determine next state	CritterState
Move entity	CritterPP
Select current target	N/A

GraphicsPlugin (abstract)

Description: Subclasses of the GraphicsPlugin class should contain information regarding the artwork of the given GameEntity. Graphics interaction type, basic image properties such as size and depth, and graceful destruction of graphics data should be contained in this class.

Justification: The graphics for a particular game entity may be animated, static, or include multiple textures. This abstract class encapsulates the basic data necessary to draw any kind of graphics from the canvas.

Classification: Component

Responsibility	Collaboration
Paint	N/A
Load content	N/A

ENTITIES

GameEntity

Description: The GameEntity class is the physical representation of an intelligent object in the game. It holds only three plugins, which together handle all state and function of a GameEntity. Any number of these plugins can be null, but most GameEntities that interact in the game world will typically contain all three.

Justification: All actors use the same components, so they are collected in one class here.

Classification: Entity

Responsibility	Collaboration
Delegate physics	PhysicsPlugin
Perform requested action	ControllerPlugin
Delegate Paint call	GraphicsPlugin

GameLevel

Description: The GameLevel class contains all the GameEntity's in a particular level.

Justification: A container class is necessary to keep references to the various entities in the game.

Classification: Model

Responsibility	Collaboration
Delegate update step to GameEntities	GameEntitiy
Simulate Farseer physics world	Farseer
Get/set global physics constants	Box2D, Farseer
Delegate paint step to Canvas	Canvas

GameEngine

Description: the GameEngine class contains the main update loop and current level.

Justification: A persistent top class is required to allow the game to change levels while maintaining state.

Responsibility	Collaboration
Contain overall game state	N/A
Simulate current level	GameLevel
Delegate Paint call	Canvas

DataParser

Description: the DataParser class reads game level files and creates a GameLevel containing the necessary state.

Justification: Need standardized way of saving and loading levels

Responsibility	Collaboration
Load/Save file in game-readable format	GameLevel

Canvas

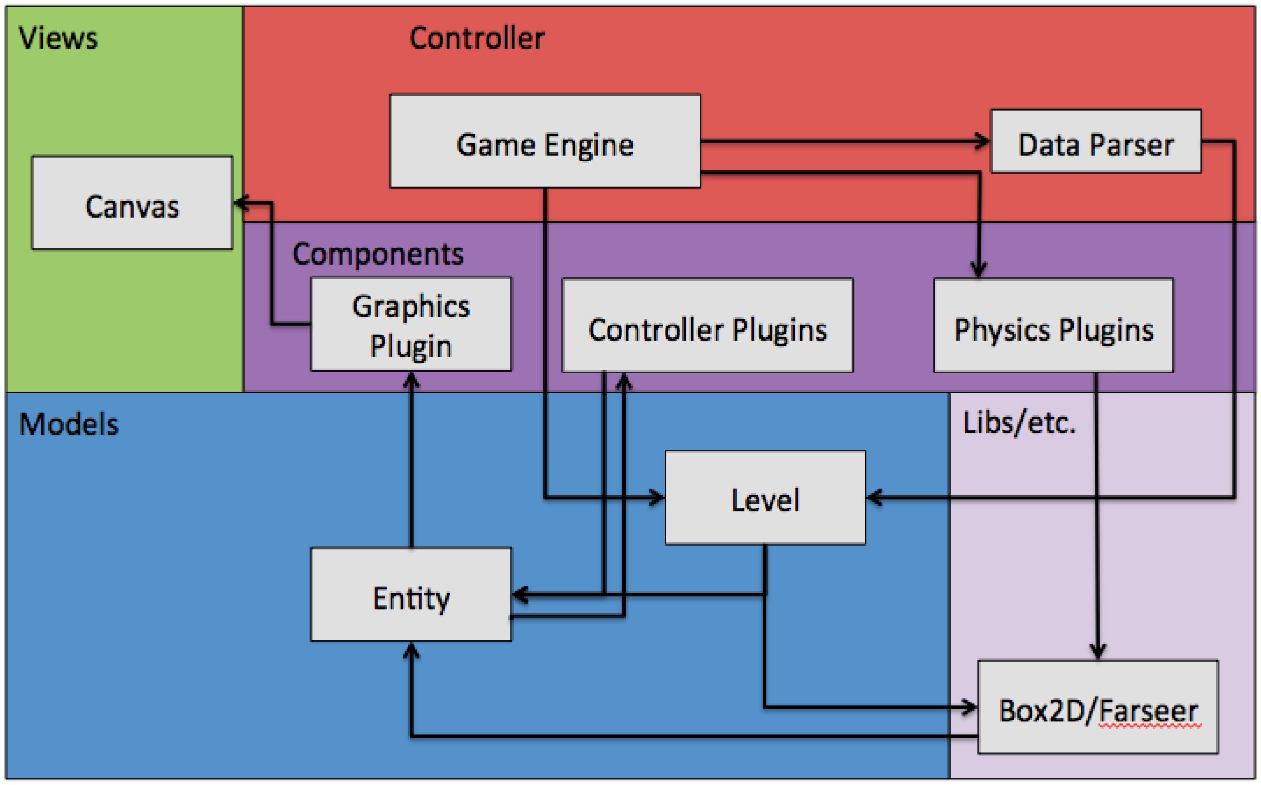
Description: The Canvas class is an abstraction of rendering, hiding away animation details

Justification: Prevents the draw method from being bloated and being in game object

Classification: View

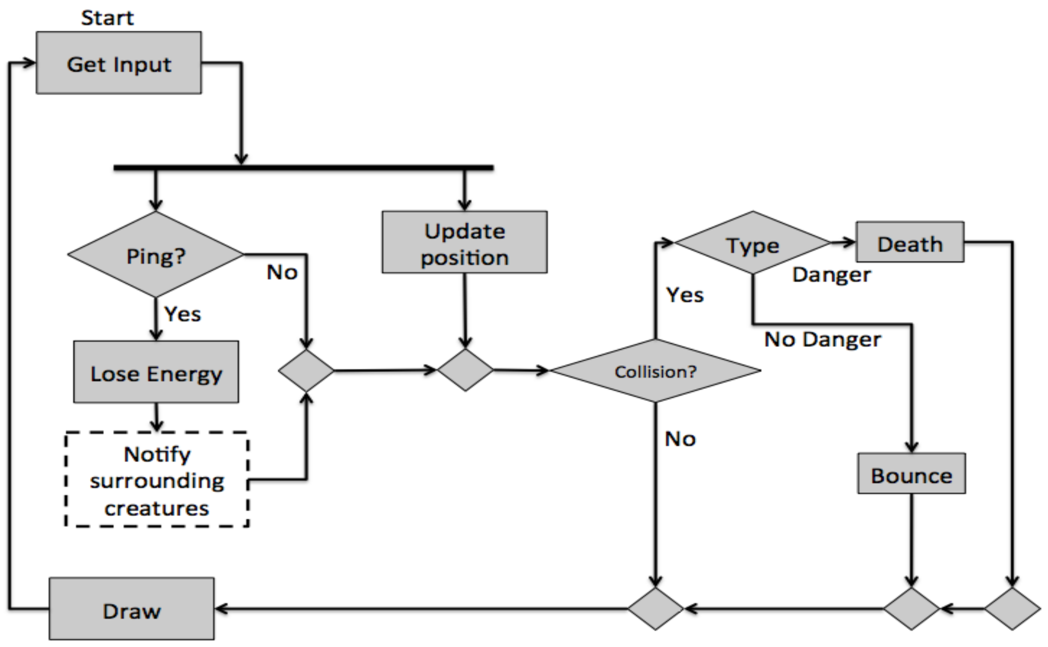
Responsibility	Collaboration
Delegate paint calls to GameEntities	GameEntity

DEPENDENCY DIAGRAM

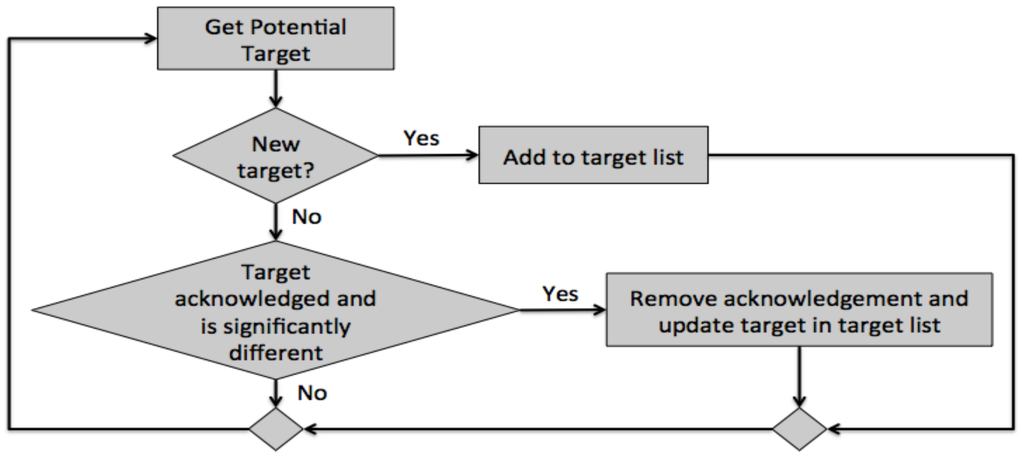


ACTIVITY DIAGRAM

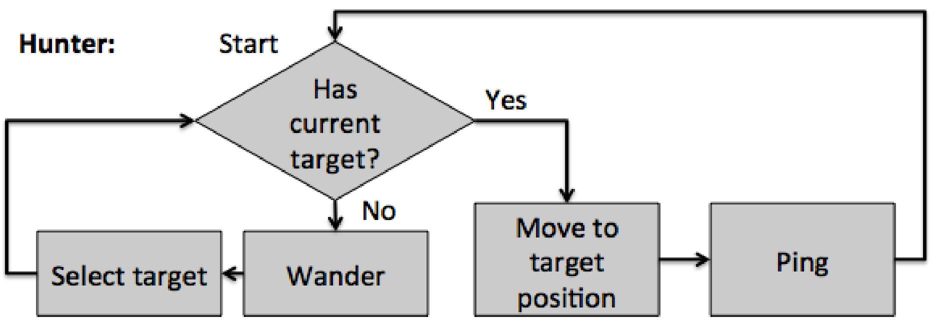
PLAYER ACTIVITY DIAGRAM

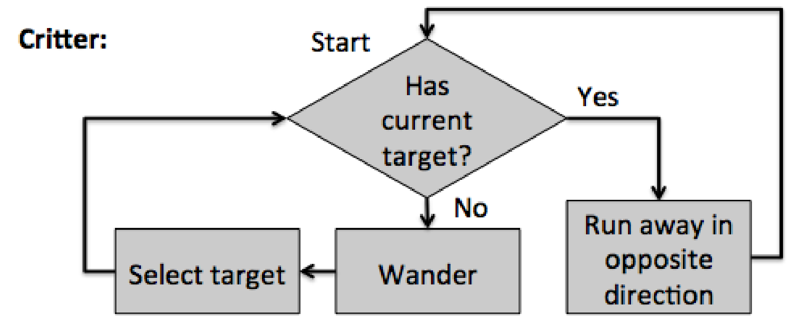


Notify surrounding creatures


Get Potential Target
New target?
Yes
Add to target list
No
Target acknowledged and is significantly different
Yes
Remove acknowledgement and update target in target list
No

AI Activity Diagram





Data Representation Model

Saved Game File

File Format: XML

Information Stored on File: We will store the list of levels completed, along with the current level.

How Information is Stored: Below is a sample saved game file. Note that you can replay and choose levels.

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<LevelsFinished currentLevel=”5”>
     <Level name=”1” state=”complete”/>
     <Level name=”2” state=”complete”/>
     <Level name=”3” state=”complete”/>
     <Level name=”4” state=”complete”/>
     <Level name=”5” state=”incomplete”/>
</LevelsFinished>
```

Level File

File Format: XML

Information Stored on File: This will store everything necessary to recreate a level. This includes size of the level, difficulty settings, components used (plugins), and assets (such as sprites).

How Information is Stored: (E) denotes an element and (A) denotes attribute

MagicNum (E)

Must be 47913277

Version (E)

The internal level maker software version

LevelSettings (E)

Contains general information about the level

- Name (E): Name of the level
- Author (E): Creator of the level
- Width (E): Integer width of the level
- Height (E): Integer height of the level

Entities (E)

Contains all entities of the level

Entity (E)

Describes one entity

- Name (A): Human-readable identification of the type of entity (unique)
- Posx (E): Integer x position
- Posy (E): Integer y position
- Plugins
    - Controller (E): The controller plugin used by the entity
    - Graphics (E): The graphics plugin used by the entity
    - Physics (E): The physics plugin used by the entity

Plugin (Controller/Graphics/Physics)

- ID (A): Unique string identifier of type
- Args (E): Contains all the information to instantiate the plugin

Arg (E)

A single parameter

- type (A): Parameter type
- [Variable] (E): Depends on type (marshalled)