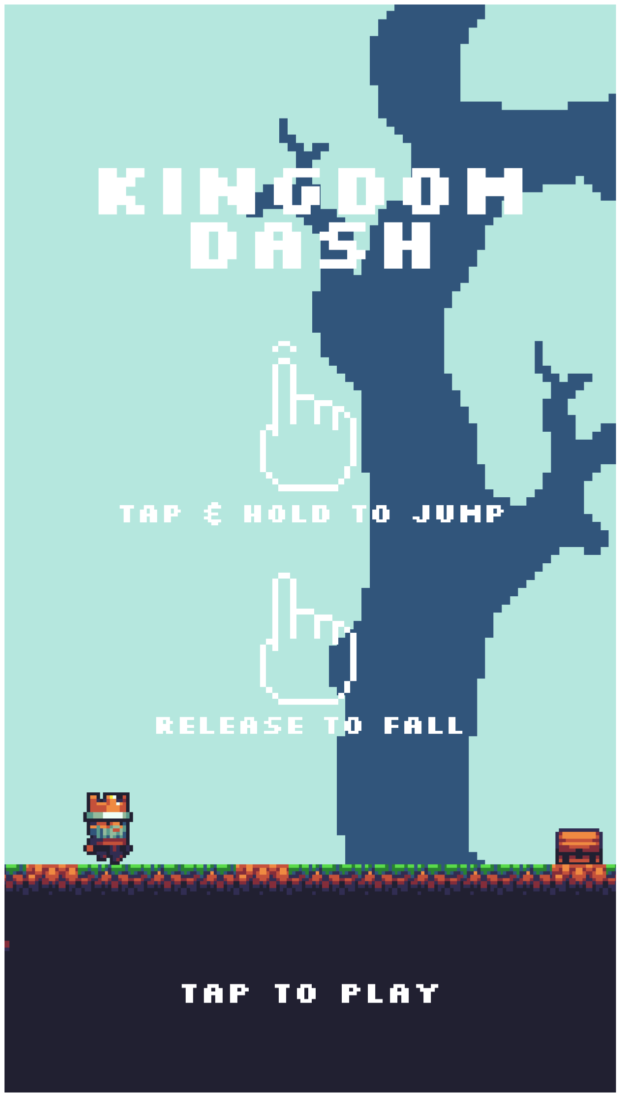


KINGDOM DASH

Technical Documentation

ABSTRACT

Kingdom Dash is a mobile arcade developed in Java. I have been a long-time hobby game developer, but this was my first finished and published video game, so it holds a special place in my projects. Previous games that I developed mostly ended in failure due to my inexperience and great ambitions as a developer, but Kingdom Dash had a very focused scope and goal, so I was able to complete it. For me, this was a rite of passage of sorts.

Jared Zhao

Table of Contents

Mission

I have been a long-time hobby game developer, beginning in the third grade through the educational programming language Terrapin Logo. My first published computer game was a number guessing game that I wrote in sixth grade. I sent the project in to the publishers of Terrapin Logo, and my project remains on their page as an example project to this day. (https://doc.terrapinlogo.com/doku.php/logolib:guess_number)

After publishing my number guessing game, I went on to develop graphical games using Java, Swing, and OpenGL. Most of my projects ended in a disastrous code base due to my inexperience as a game developer and the complicated nature of my projects, but with Kingdom Dash, I narrowed my focus and began experimenting with new ways to organize my code. With the completion of Kingdom Dash, I'm now finally fully grasping the game development process.

Design

Kingdom Dash is designed to be a simple mobile arcade. The player plays as a "King" who's goal is to remain on the platforms and go as far forward as possible. The "King" will automatically run to the right, and the player's task is to gauge when the "King" should jump. The controls are: Tap to Jump, Hold to Jump Higher, Release to Fall.

Kingdom Dash, although simple in gameplay, still contains many of the gameplay mechanics found in more complicated projects. For an example, Kingdom Dash features a fully featured 2D Physics Engine, a Terrain Generation / Degeneration System, and a Resource Management System.

The most noteworthy aspect of Kingdom Dash is perhaps its unique code organization. Unlike a typical game where the entities follow a "Tree" format where entities extend other similar entities, Kingdom Dash was designed using an Entity Component System, or ECS. (More on this in the "Code Structure" section)

Open Source

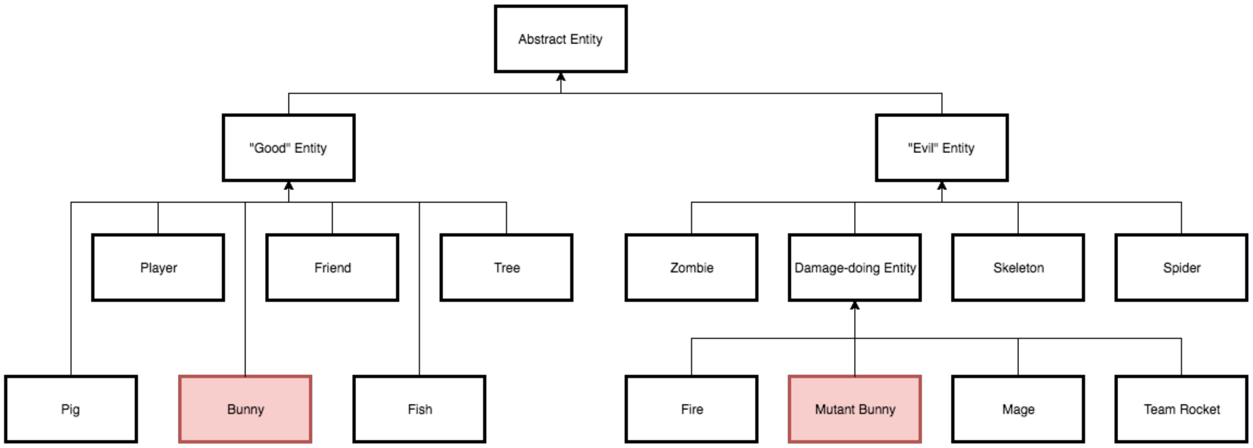
Kingdom Dash has been open sourced on Github to serve as an example to other game developers. Open source projects have been very important in my understanding of game development, and I hope Kingdom Dash can be equally valuable to other aspiring developers. (https://github.com/JahrudZ/Kingdom_Dash)

Project Structure

Unlike a traditional video game, Kingdom Dash organizes its entities in an Entity Component System, or ECS, rather than a traditional "Tree" structure like the one provided by Java's "extend" feature. This gives us a few advantages: the ability to compartmentalize entity data and game logic and the overall greater organization of the code.

A traditional "Tree" structure usually begins with a general Entity class that can be extended by other entities. All objects with the superclass, Entity, will be contained in a List which the main game loop can iterate through. In an ECS, however, Entities are composed of Components, which are just data containers. The different components that are contained within an entity will determine the entities behavior. Essentially, there is no hierarchy of Entities. Instead, there are "types" of entities.

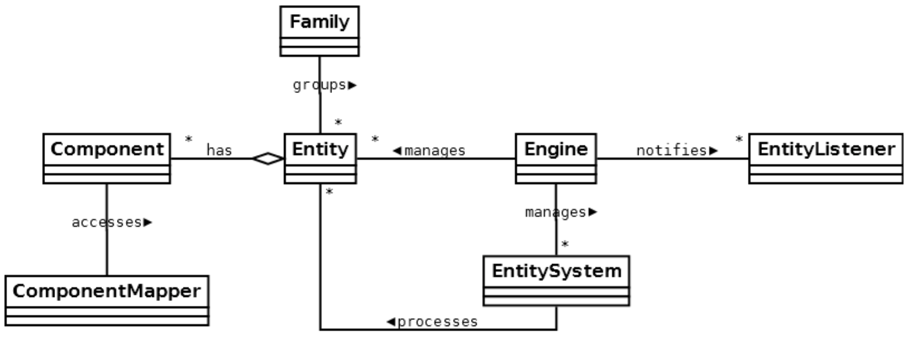
Besides its code-organizing functions, using an ECS becomes especially advantageous when entities have overlapping functionality but would end up on opposite sides of a "Tree" structure.



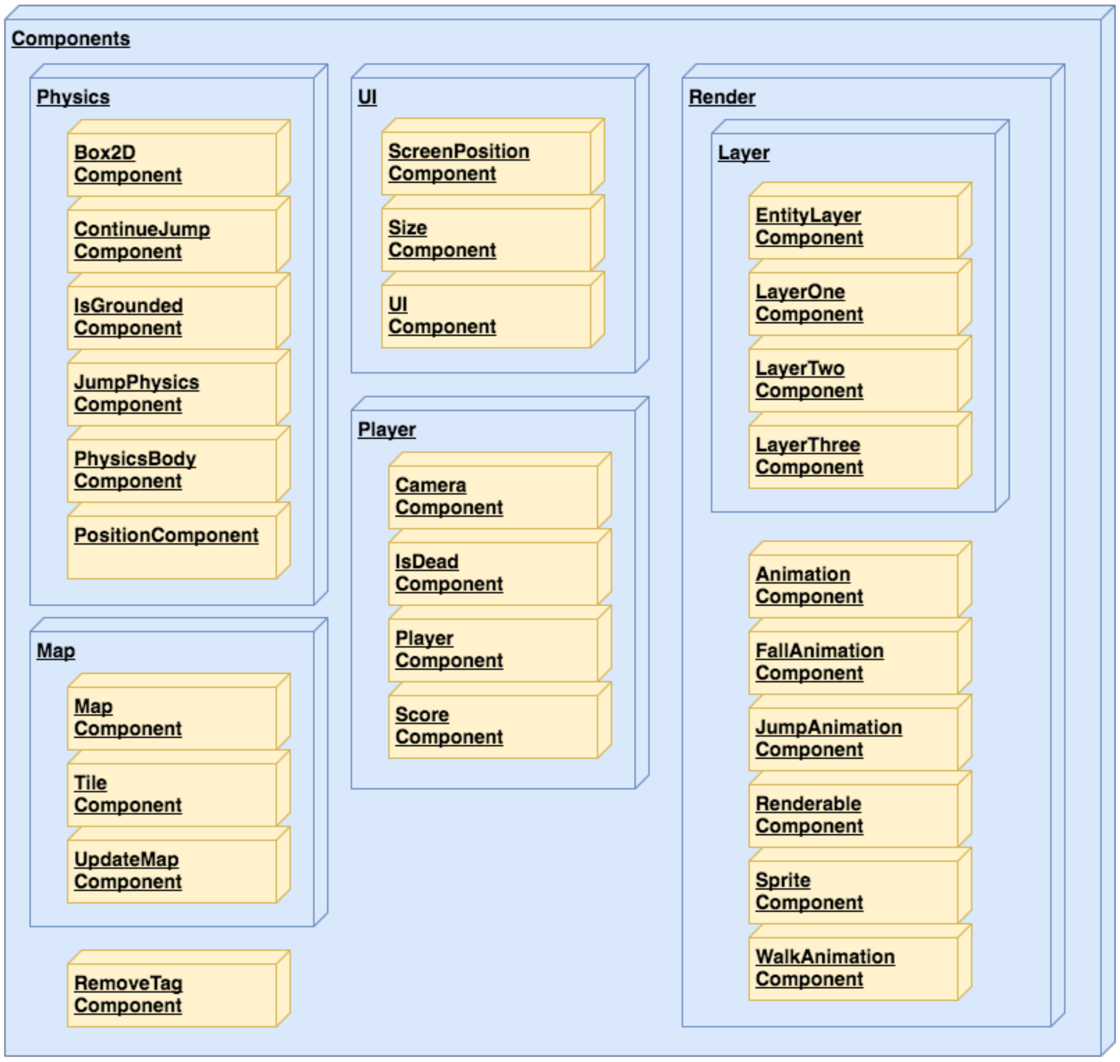
Here, the Bunny and the Mutant Bunny both require "bunny code" but are fundamentally different in function. Thus, using a "Tree" structure will require repeated code, and the project can become disorganized very quickly. Instead, with and ECS, the same functionality can be created with three components: BunnyComponent, GoodComponent, and EvilComponent. With these three components, we can still describe the functionality of the Bunny and the Mutant Bunny without the need for overcomplicated and repeated code.

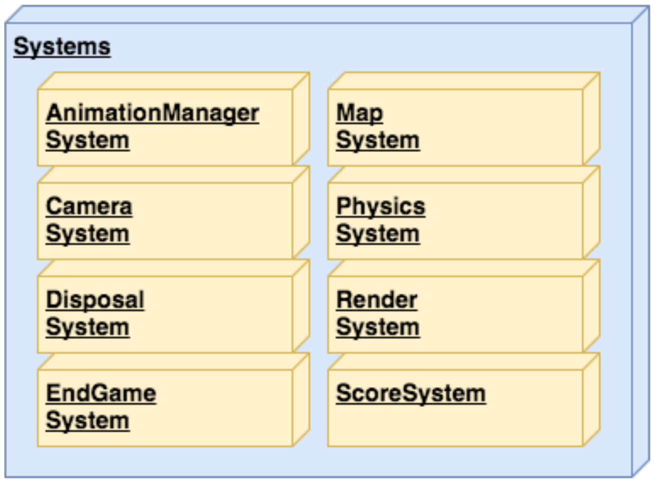
Furthermore, the game logic in a traditional game is implemented in its game loop. In an ECS, however, the game logic is implemented within its systems. Systems look for entities with the components that correlate to its function. This will allow us to compartmentalize the game logic and to isolate the code in specific systems. This will allow for easier debugging and expansion of the current game logic. For an example, instead of all game logic implemented in the game loop, we can have a separate RenderSystem, PositionSystem, CameraSystem, PhysicsSystem, etc.

The following UML diagram outlines the overall structure of an ECS. The ECS structure is meant to be applied to all aspects of game development ranging form managing entities and data to implementing game logic.



Kingdom Dash implements this ECS structure with the following components and systems. Blue represents packages and yellow represents classes. Further detail about their utilization can be found in the source code. (More on this in the "Open Source" section)





Specific Technologies

Overall Project

Kingdom Dash was developed in Java using the libGDX game development framework. libGDX allows for cross-platform development using by abstracting hardware access on the various output platforms.

Graphics

Kingdom Dash uses OpenGL 2.0 for its render calls. I did not use OpenGL 4.5 because many older Android platforms only support OpenGL 2.0.

Physics

Kingdom Dash uses the Box2D physics simulation engine for its collision detection, gravity simulation, and kinematics simulations. Box2D was originally written in C++ but was later ported to the Java language through a thin binder.

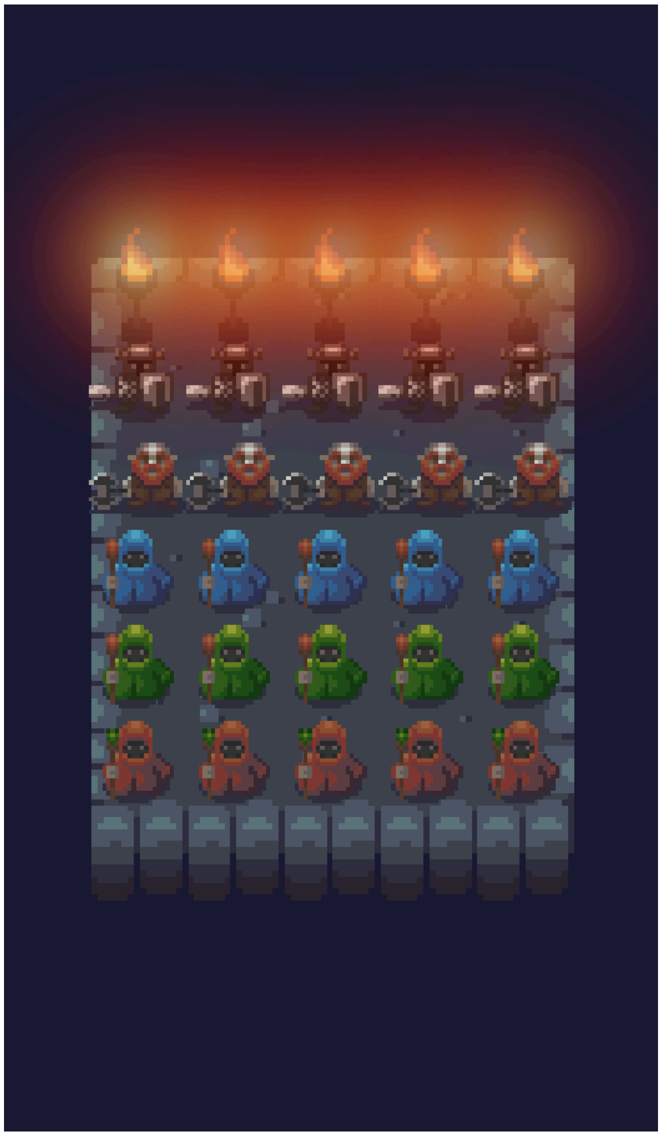
Porting to iOS

Kingdom Dash was ported to iOS through the use of RoboVM. RoboVM allows Java code to be compiled to native iOS apps. Special implementation was required to use native Java libraries.

Continued Innovation

Because I am not and do not aspire to become a professional game developer, I developed Kingdom Dash without following traditional game development conventions. Thus, I was forced to create my own creative solutions to the problems I encountered. Kingdom Dash has been an excellent exercise in creative programming, but there are still many aspects that I can improve on.

Enter, Project DungeonRogue.



While I do not aspire to become a professional game developer, I still enjoy the challenges in game development. Thus, I have begun work on a much more robust and interesting game that I have dubbed, Dungeon Rogue. In addition to the innovations in Kingdom Dash, I am implementing a dynamic audio system, multiplayer via Java Sockets, and player data stored via MySQL.