**Dung Eon : A Dungeon Crawler**

**Concept of Operations**

**COP 4331, Spring, 2016**

Team Name: Bobby Sandwich

Team Members:

* Robert Bland
* Jacob Crandall
* Jacob Jiskoot

Modification history:

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Who | Comment |
| v1.0 | 3/1/2016 | Jacob Crandall | Diagram & Notes initial |
| v1.01 | 3/4/16 | Robert Bland | Design Issues expanded |
| v1.02 | 3/4/16 | Jacob Jiskoot | Finished up Trace Through |
| ... |  |  |  |

**Contents of this Document**

Design Issues

Detailed Design Information

Trace of Requirements to Design

Design Issues

**Reusability**

* Abstract classes to allow for easy implementation of new ideas; the main abstract classes and what they contain that will allow for easy implementation are:
  + item
    - CCSprite
  + menuButton
    - Color
    - Width
    - Height
  + character
    - Health
    - Attack
    - Sprite
* The game needs to be easy to understand so that way anyone with low experience in coding can easily add or make changes to the game.

**Maintainability**

* UML lays out the architecture and the relationships between the system components
  + This allows us to visually see dependencies when creating or editing any classes
* Documentation is key for maintaining the game, so having a changelog for other group members to see as well as another changelog for customers to see would help us generate more feedback
  + Updating our edits and our changes to a change log will tell other group members what new dependency is added to avoid possible conflicts
  + Allows customers to assist in bug tracking once the game launches
  + Good comments for code will allow us to easily assist each other in case someone struggle in the group
  + Allows us to possibly let members of the gaming community to easily edit our possible final version of the game

**Testability**

* Testing will happen at every iteration of each new prototype
  + Main purpose is to check if the new game mechanics work with existing game mechanics
  + Another purpose is to try and work out some of the bugs that appear and mark them down for someone to fix
  + Also to check the performance and response times in case an implemented algorithm bogs down the system
  + Final prototype will be tested more extensively than the other prototypes as that is the one that the customers will be using

**Performance**

* Response time is crucial to keep a fluid game
  + If the user’s input does not receive output within one second then response time is too slow
  + Terrible response times could keep the player from enjoying the game since it will feel unresponsive
* Performance shouldn’t be terrible from the game since calculations will be rather small
  + All algorithms that are performed, however, should not have a time complexity that exceeds n^2

**Portability**

* Manageable and possible to deliver to both Android and iOS
  + Possible since CoCosSharp only requires a few lines of code to swap between different operating systems
* Availability will first be for Androids

**Safety**

* Application has no internet so we are not worried about internet security
* Application has no credential information saved, so no encryptions are needed
* Application won’t require enough processing power overheat or cause the phone to brick, so not much needs to be done in terms of safety

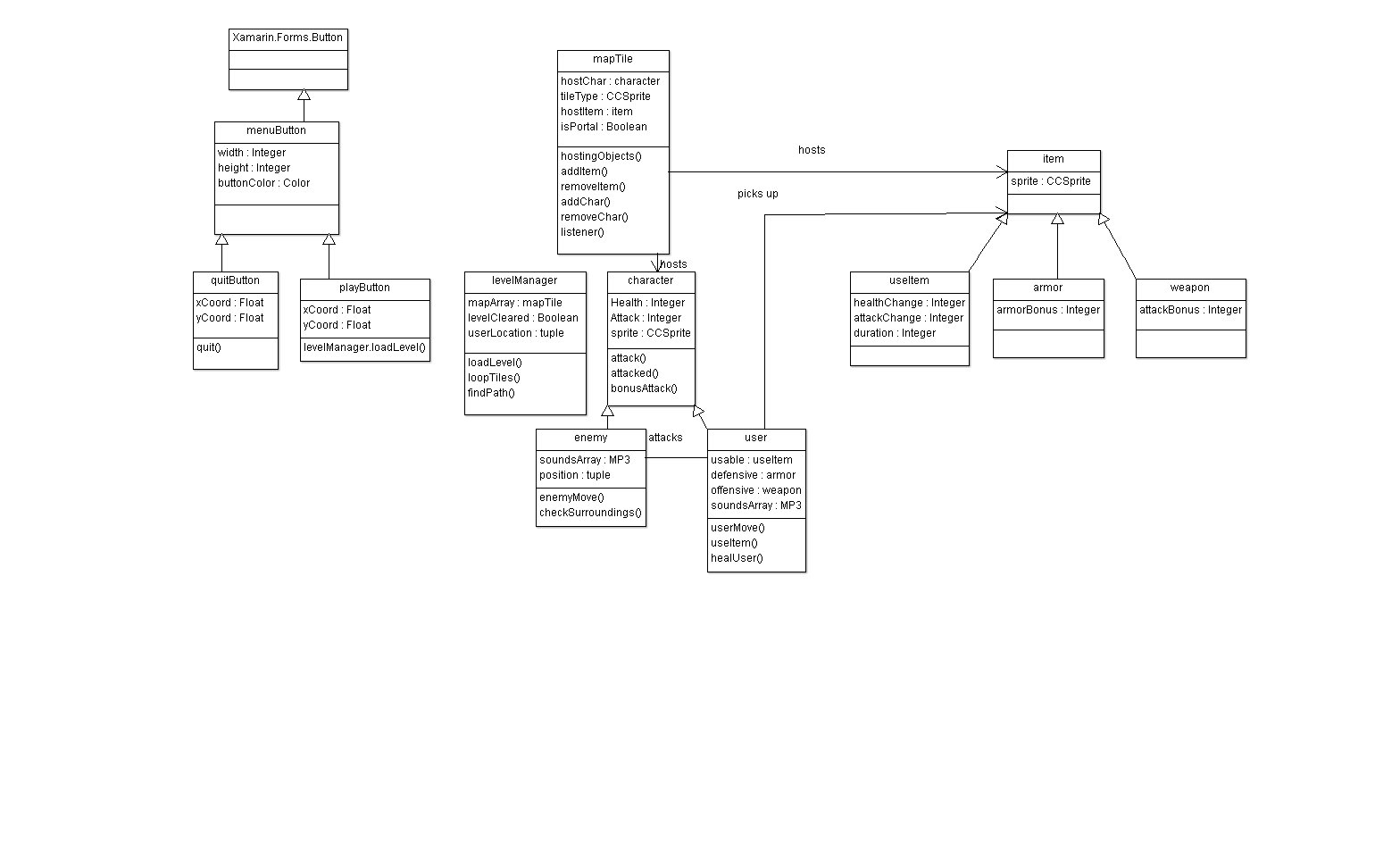
**Prototyping**

* Key part of our development and life cycle
* A new prototype is developed for every implementation
  + Purpose previously described
  + Allows us to also try out other ideas in case we don’t like new prototypes
* An initial prototype will consist of:
  + Basic controls
  + Main menu
  + A way to start the game
  + A way to quit out of the game

As we are going into the implementation stages we will begin by creating core prototypes. We hope to build off of these prototypes and use them in our final product. However, should we encounter issues with our prototypes and ideas that were based on these prototypes we will document those issues here. Along with those issues we will document the proposed solution to the problem found in the current prototype. This will set us up to create a new prototype and we will repeat the process.

Upon the creation of our first prototype we will populate this area with the initial information.

Detailed Design Information



Utilizing diagrams from the High-Level :

**CocoSharp / Application :** This is the main() of our program. It loads up the initial settings for android via CocoSharp. It will create two buttons, a quitButton and a playButton. On click quit will quit the application utilizing the CCs quit command. On click play will create and load a new instance of levelManager.

**Level Manager :** loadLevel() - this will be called upon creating a new level manager. This creates an array of map tiles (mapArray) and shall use CCs and coords provided by the map tiles to display. The level creation algorithm will have counters : 1 end portal, 1-4 enemies, 1 user, 0-2 items. It will place this many of each object throughout the map randomly by using the add functions attached to maptiles. The maptiles will be placed by using an algorithm that takes an initial tile in the middle of the array. It will then randomly choose up,down,left, right and add a tile in this direction. Findpath utilizes dikstra’s and calls a set of userMoves upon finding a path.

**Map Tiles :** Hosting objects simply loops through each variable it owns and returns a tuple of these objects. Adding functions simply set the respective variable to a passed in variable. Remove functions set that variable to null.

Listener() utilizes xamarin’s onClickListener and defines actions here. Should it have an item it will call add item to the char and remove item from itself. Should it have an enemy it will move the char and call attack functions. Should it have nothing it will simply move the user.

**Characters -** Characters functions simply interact with their variables by setting them. Attack searches nearby maptiles in the array based on its coordinates and then sets their health to - the attack value. Attacked is an interface for others to call. So attacked takes an attack integer and subtracts it. Bonusattack() takes a character's attack and adds to that variable.

**Enemy -** Enemy move simply chooses an adjacent tile randomly (that is available so check that this tile exists) then moves to it if possible. checkSurroundings simply checks each adjacent tile to see if a user is there, if so it calls attack functions on it.

**User -** User move is similar to enemy move but not random. A full path will be established by levelmanager’s findPath call. userHeal will be called every 2 seconds and simply call the userAttacked on itself with a negative value, this simulates healing. In userHeal check that the health does not go over a defined constant MAXHEALTH + armor.Armorbonus. In useItem check the item type. If a use item and useItem.attackChange == 0 call attack on yourself with a negative value defined by the item. If a useItem and and useItem.healthChange == 0 call user’s bonusAttack function to add to attack.

|  |  |  |
| --- | --- | --- |
| Requirement | Document | Trace |
| MenuReq | UML | Utilizing the Xamarin built in buttons we show buttons with attached functionality in the UML class diagram. |
| MapReq | UML | user object has a function for moving the char - mapTiles support this with add and remove char  mapTile supports the ability for users picking up items  mapTiles support hosting enemies and enemies support being attacked and attacking  levelManager will check the requirements if a level is complete and load a new level should it need to |
| ItemReq | UML | mapTile supports equipping a user with an item with addItem()  user supports using an item with useItem() |
| PlayReq | UML | We have instances of menuButtons in play and quit buttons.  They have attached functionality quit() and loadLevel() specifically |
| ControlReq | UML | Maptiles have listeners that will be checked in each loop of the game manager’s update loop - checked in our main() |
| SoundReq | UML | Characters provide sound files and we will use CocoSharps built in sound engine to play sounds |
| INTR1 | UML | mapTiles provide the hostingObjects() function which returns all objects that it is hosting |
| INTR2 | UML | mapTile objects have variables to hold these objects as well as Add and remove functions to interact with them  It also holds a CCsprite to display images |
| INTR3 | UML | character objects may utilize the attack and attacked functionality to interact with health  there is a bonusAttack() functionality to interact with attack stats |
| INTR4 | UML | users provide a useItem() function, enemies will be unable to use items useItem() simply interacts with a characters stats in a defined way by the specific use item |
| INTR5 | UML | enemy type characters provide a checkSurroundings() function that shall check each surrounding tile based on its position, if that tile contains a char then attack it |
| AOS | NA | The game is created for the purpose of running on android. |
| HF1 | UML | The listeners shall be connected to define behavior, so behavior that is not being listened for does not have defined behavior. This is difficult to trace but still satisfied. |
| OnlineDoc | N/A | See - <https://github.com/jacobmcrandall/POOPCrawler> - documentation aggregated here |
| DARLVL | UML | The levelManager class constructor shall create an array of mapTiles that will all be CONNECTED (creating a path) and placing one end-portal - thus allowing for the map to be winnable |
| DARDIJK | UML | levelManager employs a findPath() algorithm. This creates an array of coordinates that it calls on the userMove to get it to move efficiently |
| DARHEAL | UML | User has an algorithm for heal(). This replenishes the user’s health over a period of time. |
| XamCoReq | N/A | The game will be created with certain libraries supposed to simplify coding for phone devices: Xamarin, CocosSharp |
| FundsReq | N/A | No real funds are required for this project but we would like sprites |
| SpaceReq | N/A | Can’t be represented in the UML. We will use efficient and non wasteful coding standards |
| ProcessReq | N/A | Can’t be represented in the UML. We will work towards following our coding standards to meet this requirement |
| TestReq | N/A | Can’t be represented in the UML. We will unit test all of our code to prevent as many errors as we can. |
| BugReq | N/A | We will use appropriate error reporting in our code in order to help both us as developers know what to fix and the user know what to not repeat. |
| RelReq | N/A | We will play the game ourselves to check for potential bugs in our code as well as the code of our partners |
| AvailReq | N/A | We will try to publish the app to Android app stores with a value of free |