5-21-2014

Mobile Game Application: The MagicTale

Lu Cao
Santa Clara University

Albert Chang
Santa Clara University

Yetian Mao
Santa Clara University

Santa Clara University
DEPARTMENT of COMPUTER ENGINEERING

Date: June 3, 2014

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Albert Chang, Lu Cao, Yetian Mao

ENTITLED

Mobile Game Application: The MagicTale

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THESIS ADVISOR

DEPARTMENTCHAIR

Mobile Game Application: The MagicTale

by:

Albert Chang, Lu Cao, Yetian Mao

SENIOR DESIGN DESIGN REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California

May 21, 2014

1. Abstract

Many smartphone games today consist of players making simple and repetitive actions with their fingers to play the game. Compared to traditional PC Role Playing games (RPG), current mobile games do not provide the same fictional aspects and complex components. Players' creativity is almost non-existent in most simple mobile application games. To address this issue, we propose to create a IOS mobile application game which combines the complex components of traditional PC RPGs and touch screen actions of mobile games.

2. Acknowledgments

The MagicTale team would like to thank Dr. Maria Pantoja, our senior design advisor, for helping and working alongside us during the course of our project. We would also like to thank all the students that participated in testing our mobile game. With the support of these individuals, we were able to create a functional, user-friendly, and appealing mobile game.

3. Table of Contents

1. Abstract....2
2. Acknowledgments....2
3. Table of Contents....3
4. Table of Figures....4
5. Introduction....5
    5.1: Problem Statement....5
6. Main Body....6
    6.1: Project Requirements....6
    6.2: Use Cases....7
    6.3: Project Design....10
    6.4: Project Visual Design....12
    6.5: Technology Used....17
    6.6: Societal Issues....18
    6.7: Test Plan....20
    6.8: Risk Analysis....21
    6.9: Development Timeline....22
7. Conclusions....23
8. Appendix....24

4. Table of Figures

Figure 6.4.1: The MagicTale Application Login Screen.....13
Figure 6.4.2: Stage Selection Screen.....13
Figure 6.4.3: Entering the Stage.....14
Figure 6.4.4: Fireball Animation.....14
Figure 6.4.5: Magic Circle with Different Attributes on Each Node.....15
Figure 6.4.6: Iceball Animation.....15
Figure 6.4.7: Piercing Ice Shard Animation.....16
Figure 6.4.8: Enemy Minions Attacking Player.....16
Figure 6.9.1: Development Timeline.....21

5. Introduction

5.1: Problem Statement

Many people seek entertainment through games on their smartphones. However, many games on the current smartphone platforms involve the players performing mindless and repetitive actions to achieve their goals. Traditional PC Role Playing games (RPG) provide players with a variety of heroic fantasy character roles allowing players to use their imagination and creativity to create a unique character. Unfortunately, smartphone games today not only lack positive influences on the players, but also restrict players' creativity.

We designed an RPG action game that will combine the complexity of the traditional PC games and touch screen actions of the smartphone games on the iOS platform. This game consists of a fantasy world where characters inflict damage by casting their magic spells and skill sets. Players can cast different magic spells by drawing different geometric shapes in the magic circle on the phone screen. Each spell has its own attribute corresponding to a certain shape on the magic circle. For instance, drawing a circle on the magic circle would trigger a fire attribute spell. In order to cast advanced spells, players need to have a thorough understanding of the magic circle system along with decent geometry drawing skills. Players will gradually and progressively learn many different geometric shapes, which represent spells with different attributes. By combining multiple spells, meaning a player draws two or more different spells on the same magic circle, a player can generate a new spell. When players combine spells, they need to understand how to solve questions such as how many triangles can be drawn in a pentagon. This skill system would allow players to fully use their imagination to generate different kinds of spells. Additionally, by combining game items and skills, the game would offer players more options and possibilities to explore. Finally, with a positive and heroic storyline, the game may offer a positive influence on young players. Overall, this creates an open game environment for players to create their own unique character based on their infinite imagination while gaining knowledge of geometry.

6. Main Body

6.1: Project Requirements

Before developing our mobile application, we created a list of functional and non-functional requirements. Functional requirements define the behavior of the system. Non-functional requirements define the system's performance characteristics. Critical requirements are requirements which are most important and necessary to the project. Recommended requirements are important but not an absolute necessity. Suggested requirements would add to the project, but are the least degree of importance out of the three.

Functional Requirements

Requirement	*Degree of Importance*
System must allow users to save their current game progress.	Critical
System must allow users to start a new game.	Critical
System must allow users to modify game settings.	Critical
System must allow users to interact with other users.	Recommended
System must support a large capacity of online users.	Recommended
System must allow users to access a help section for additional information about the application.	Recommended
System must have in-app purchase.	Critical

Non-Functional Requirements

Requirement	*Degree of Importance*
System must be be user-friendly and accessible by users.	Critical
System must be acceptable and available in many countries.	Critical
System must be secure and reliable to users.	Critical

6.2: Use Cases

Use cases list the actor using the application, their end goal, preconditions, postconditions, the scenario to complete the form, and any exceptions they may encounter.

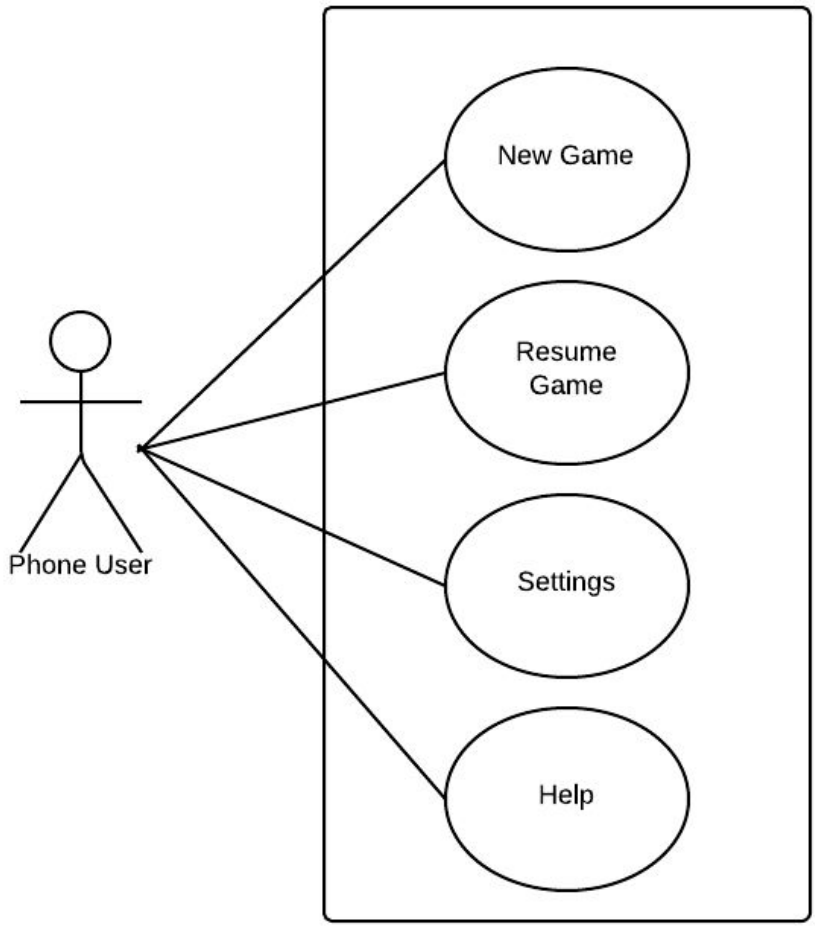


Figure 6.2.1: Use Case Diagram for interaction between user and user interface.

Use Case #1: Create a spell

Actor: User

Goal: Create a new spell

Preconditions:

- The specific combination of magic attributes has never been used
- User has fulfilled the game requirement to obtain an additional spell

Postconditions:

- The new magic spell is created

Scenario:

1. User creates a new spell by drawing on the magic circle
2. User learns a new spell from an enemy minon

Use Case #2: Add a Magic spell

Actor: User

Goal: Add a magic spell to the character spellbook

Preconditions:

- The magic spell has been drawn and used once
- User has fulfilled the game requirement to obtain a spellbook

Postconditions:

- The new magic spell is added to the character spellbook

Scenario:

1. User creates a new spell by drawing on the magic circle
2. User acquires a new spell from an enemy minion

Use Case #3: Cast a Magic spell

Actor: User

Goal: Cast a magic spell

Preconditions:

- A valid magic spell has been drawn on the magic circle
- User has fulfilled the game requirement to own the spell

Postconditions:

- The user casts magic spell

Scenario:

1. User casts a spell at an enemy
2. User casts a spell on an object
3. User casts a spell on himself/herself

Use Case #4: Level up

Actor: User

Goal: Level up

Preconditions:

- User has gained enough experience

Postconditions:

- User's level goes up by one

Scenario:

1. User gains experience from killing an enemy
2. User levels up after killing an enemy
3. User levels up after using an item
4. User levels up after accomplishing a quest

Use Case #5: Pick up Magic item drop

Actor: User

Goal: Obtain a magic item

Preconditions:

- User has killed a monster or an enemy minion
- User's level is within the range of the map level requirement

Postconditions:

- Magic item appears in user's inventory

Scenario:

1. User kills a monster or an enemy minion
2. Enemy monster drops random magic items when killed
3. User picks up magic item

Use Case #6: Pass a stage

Actor: User

Goal: Pass a stage

Preconditions:

- User has killed the boss in the stage and all minions
- User must have completed the quests within the stage

Postconditions:

- User has access to a new stage and can continue on with the story

Scenario:

1. User enters the desired stage.
2. User completes all the required quests within the stage.
3. User defeats the boss in a specific stage
4. User has access to a new stage

6.3: Project Design

The game can be divided into three main components: action battle, magic attributes, and magic items. In general, the game is stage-based, meaning that players need to clear the first stage before moving onto the second stage. The game scene is in 2D and it is displayed to players from a third person perspective. While playing the game, players can only see a portion of the map, but they can explore the rest of the map.

Action Battle

The action battle in the game is based upon the feature that players cast magic spells by drawing magic metrics on the phone screen. The basic idea of action battle in this game is that players need to quickly and precisely draw a shape in the magic circle to cast a spell while keeping a safe distance between the monsters and their in-game characters. More specifically, first players need to run away from the monster by tapping on a safe area on the current portion of the map. While the character is running away from the monster, players need to finish drawing the magic circle on the screen. Since the amount of time to draw the magic circle is relatively short and multitasking is required while drawing the matrix, the game is highly action based. Depending on the accuracy of the drawings, the damage of the spell varies. The more precise the drawing is, the more damage the spell inflicts. The time spent on drawing the matrix also affects the damage of the spell. The less time spent on drawing the matrix, the more damage the spell inflicts. After the matrix is completed, the character raises its wand to indicate that a spell is ready. Players simply tap a spot on the screen to launch the spell towards that location.

Magic Attributes

By drawing different shapes in the magic circle, it triggers magic spells with different attributes. We designed seven attributes total, which are fire, ice, water, poison, lightning, wind, sand. Players can combine magic attributes in any way they wish. To do so, they can simply draw two or more shapes on the same matrix to create a new spell. Among the seven attributes, some compatible attributes can be combined to create a stronger spell, such as wind and fire, since wind enhances fire. However, some incompatible attributes should not be combined together, such as sand and fire, since sand extinguishes fire.

Magic Item

Items drop when monsters are killed. Magic items have different rankings based upon their rarity. The rankings are normal item, superior item and rare item. In general, the higher the rank of the magic item is, the more powerful the item is. The rarer an item is, the less chance there is for it to drop. A whole set of magic items include weapon, armor, helmet, gloves, rings, and boots. Each item mainly enhances a unique attribute. For example, boots mainly enhance the character's movement speed and gloves improve the character's attack speed. Higher ranking items, such as rare items, may enhance more than one attribute. The magic item also has a level requirement. The higher the level requirement is, in general, the better the item is.

By combining the magic items and magic attributes, players are free to create a wizard with a focus on any kind of attribute. For example, a fire wizard is a wizard that has skill up points on the fire attribute and inflicts mainly fire damage. The players can create their own spell by combining different spells. The game provides players an environment to fully use their imagination.

6.4: Project Visual Design

Figure 6.4.1: The MagicTale Application Login Screen

Figure 6.4.2: Stage Selection Screen

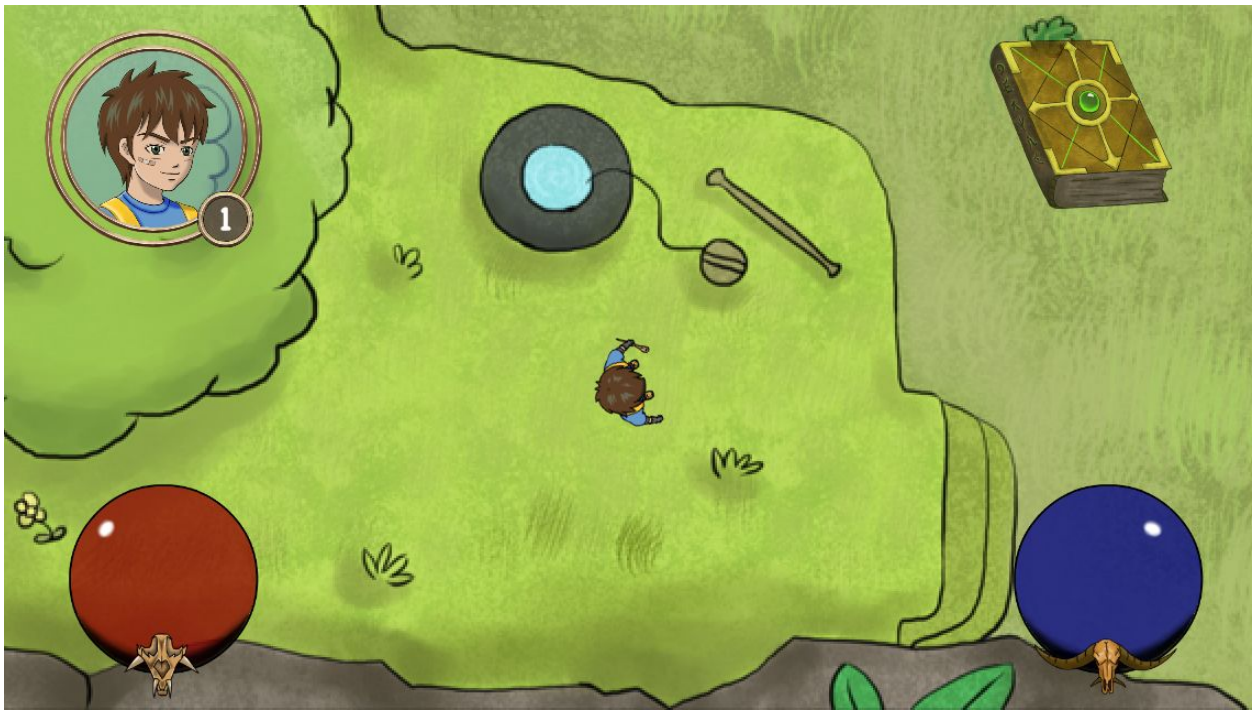


Figure 6.4.3: Entering the Stage

Figure 6.4.4: Fireball Animation

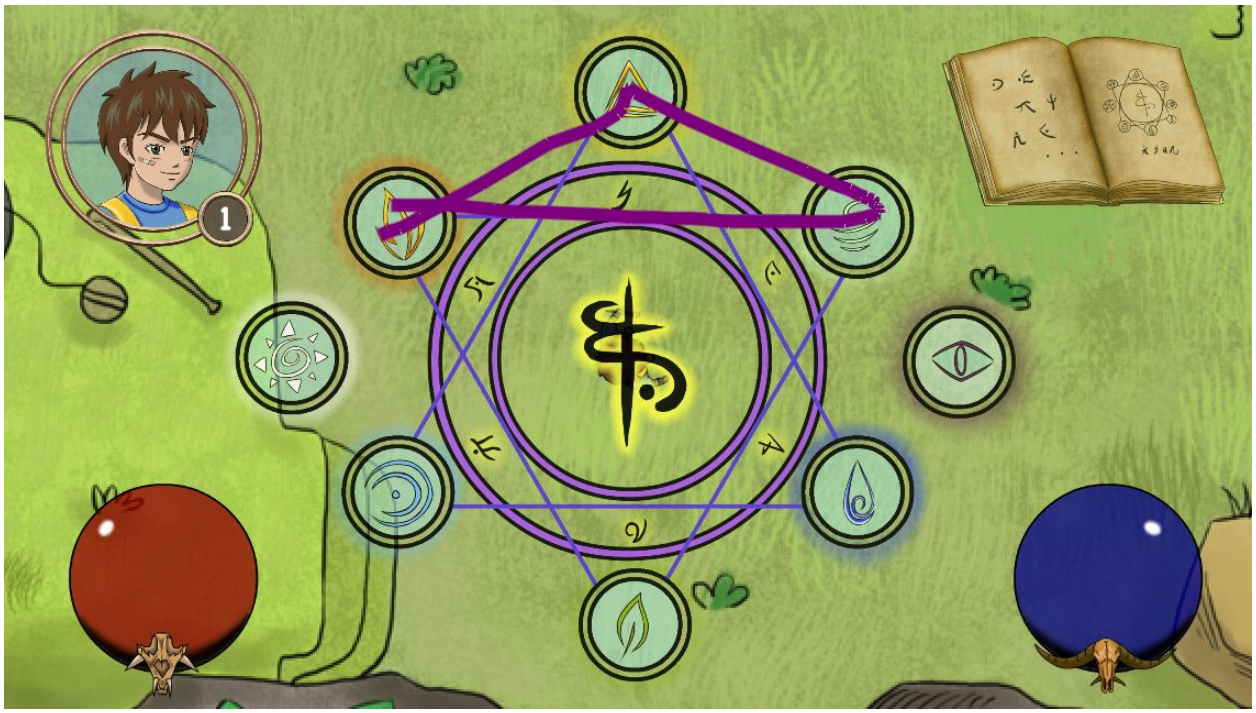


Figure 6.4.5: Magic Circle with Different Attributes on Each Node

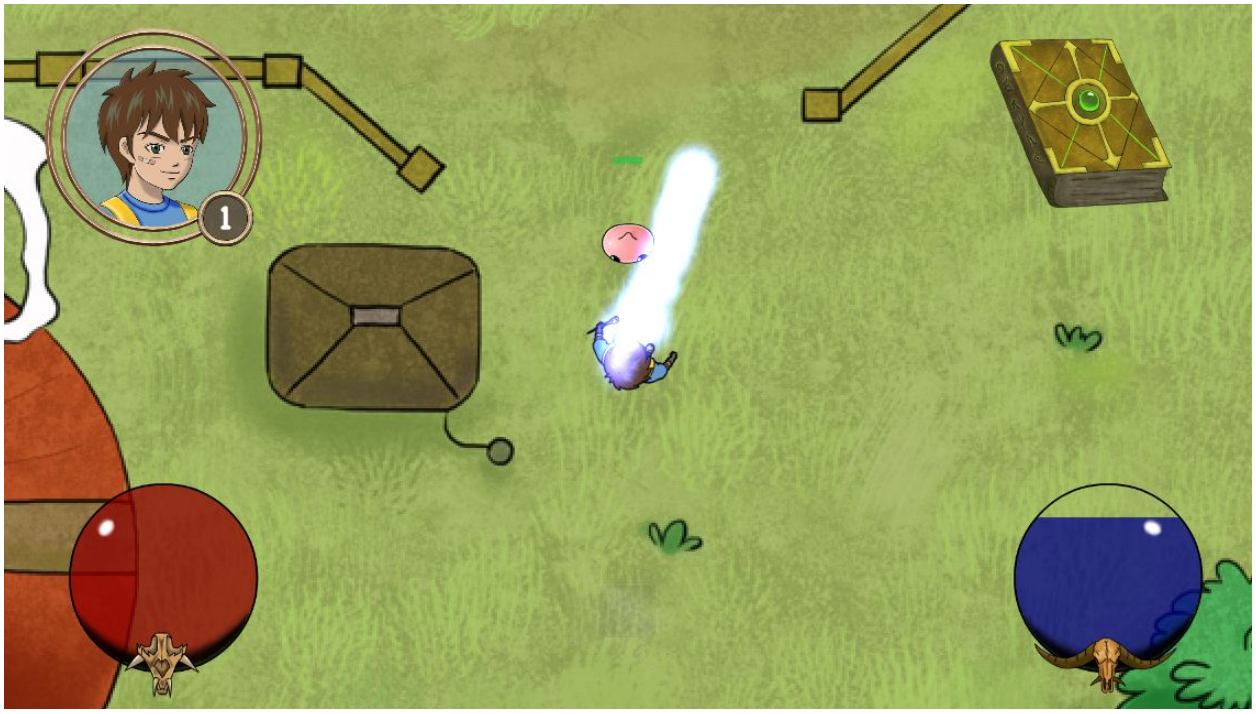


Figure 6.4.6: Iceball Animation

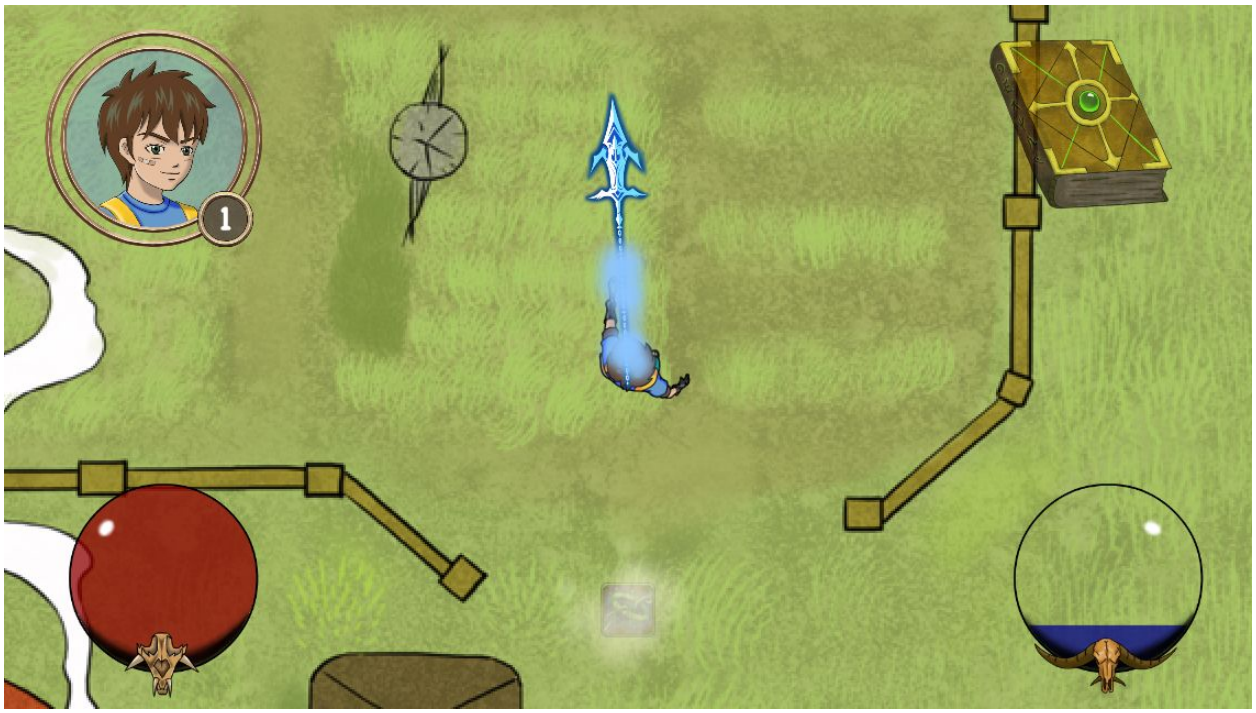


Figure 6.4.7: Piercing Ice Shard Animation

Figure 6.4.8: Enemy Minions Attacking Player

6.5: Technology Used

The app is primarily implemented through Xcode. Because IOS by far has the highest amount of users who are willing to pay for their apps, we chose the IOS over other mobile platforms. Xcode is a very powerful tool. It includes features such as IOS simulators and databases. We chose the Sprite Kit as our primary development environment. Sprite Kit is a game engine natively supported by Apple that is mainly used for 2D game design. Because the Sprite Kit is provided by Apple, this game engine must be compatible with the current iOS system and must be able to be supported by Apple in the future. The Sprite Kit is relatively easy to learn and it also provides enough functionality that we needed for the design of this project.

For data sharing among group members, we choose to use Dropbox and Github. Github has a great version control system, which allows us to keep track of different versions of our game. Dropbox, on the other hand, offers a very handy way to quickly share a file or resources.

6.6: Societal Issues

Regarding ethical concerns, our mobile application does not encounter many of these issues. Our application does not require users to enter any personal information. Our mobile game does not have a centralized server. Therefore, hackers cannot steal a players' game data because the data is stored locally.

Socially, our mobile game would assist in bringing mobile gamers closer together. With our fantasy world storyline in our mobile application game, players are able to compete against other players to see who can defeat the many boss stages and reach the farthest in the game. Our mobile game allows for players to develop their creativity by creating and using various spell attributes within the game. Because geometric knowledge is essential for players to advance in our game. We design the game in such way that players can easily obtain geometric knowledge by simple playing the game.

Economically, our mobile application does not require a centralized server to run. We do not have to take in account for server costs. To create revenue, we could implement ad banners in the starting menu of our mobile game, which would not change the gameplay.

Regarding health and safety, our mobile application does not encounter many of these issues. The main concern regarding health would be users spending hours playing the game and staring at a mobile screen for hours straight. One way to remedy this issue is to have a pop up message that encourages users to take breaks every couple hours when playing our mobile game.

Manufacturing our mobile game would be a relatively easy and simple process. With an already developed mobile platform such as the IOS app store, our mobile game could be easily distributed to users and allow us to provide customers with updates and bug fixes.

Sustainability was taken into consideration when developing our mobile application. We aimed to create a mobile game that was not only easy to use, but also easy for change. We designed the MagicTale with the ability to provide players with many stages. With our application built on the IOS platform, we are able to provide players with updates and new stages through the IOS App store.

Environment impact of our mobile game is fairly low due to the mobile game based on software architecture versus hardware architecture. Maintenance of our mobile game is fairly low because we do not use a database or have a centralized server dedicated to the game. One way we can reduce negative environmental impact is by writing efficient code to make sure battery life is not wastefully used.

Usability of our mobile game is straightforward for users. The user interface of our mobile application is designed for simplicity over complexity. The placement of the magic circle and magic book allow for players to easy navigate their character around the map. One slight concern is our mobile game is based on the Role-Playing Game (RPG) genre. RPG genre is different from other game genres and can take players time adjusting to.

Lifelong Learning has been acquired by our team during the development of our mobile game. Our team went through all the stages of software development process from forming the critical requirements to devising our test plan. We also learned to work

together as a team and meet project deadlines. With undergoing the software development process as a team, we can carry this essential experience as we work for large software companies.

Compassion played a role in developing our mobile application. As a team, we want to bring mobile gamers closer together as well expand the mobile gaming community by developing a mobile game.

Political Issues do not exist within our mobile application. We designed the MagicTale for the purpose to providing players with a more complex mobile application game. We designed the mobile game to be functional and user-friendly.

6.7: Test Plan

The majority of testing our application occurred after the requirements were formalized and during all stages of the coding process.

Iphone Simulator

To test functionality and features of our iphone application, we decided to use Xcode's Iphone Simulator. Xcode's Iphone Simulator allows us to open the iPhone interface in a window on a Mac computer. With iOS simulator, we can test and emulate various finger taps, finger drags, device rotations, and other user actions using the mouse and keyboard.

Unit Testing

Our team created unit tests during the development process to ensure the code meets project design and behaved as needed. We tested each unit of the system individually to make sure the unit returns the correct output in our test cases.

Prototypes

Prototypes of the mobile application and user interface allowed us to create mockups of what the system will look like. Prototypes allowed us to test the user-friendliness of the system without spending large amounts of time and effort during the actual development of the system.

Acceptance Testing

We incorporated acceptance testing after testing our back-end functionality. had in-house testing between team members during winter quarter and prepared the system for beta testing in the spring quarter. We had students from various clubs play our mobile game to provide our team with feedback from an actual user's perspective.

6.8: Risk Analysis

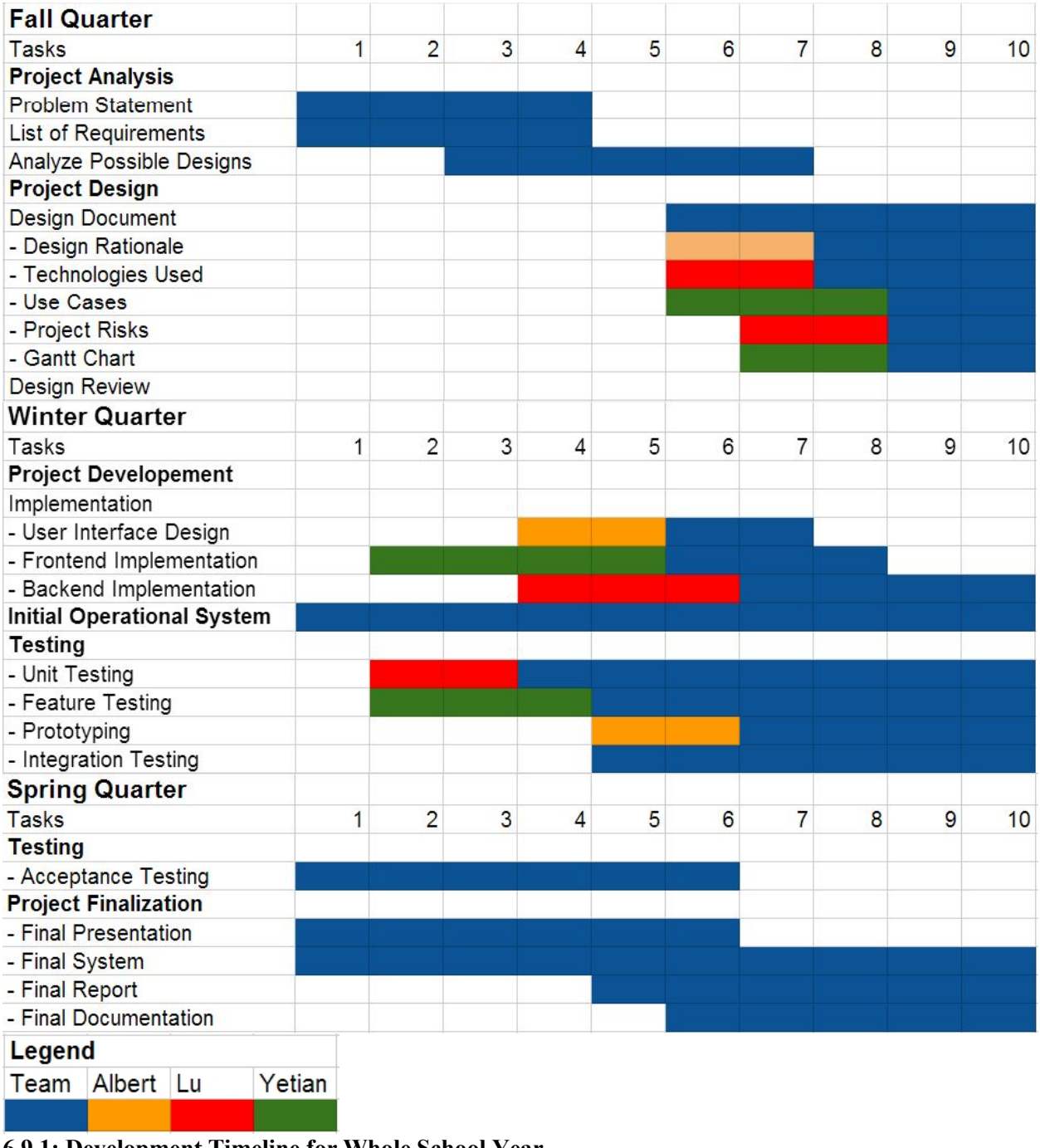
The risk analysis table displays the possible problems that the team might have encountered while developing and implementing the system. The table shows the probability of each risk occurring, the severity of each risk, and the impact the risk would have on the development of our system. The table also shows mitigation strategies to solve the problems.

Risk	Consequence	P	S	I	Mitigation Strategy
Illness or Emergency	Team may fall behind on work	1	5	5	• Assign a coordinator to each component of project • Schedule extra meetings to work
New Technologies	Learning new technologies can delay completion	0.8	4	3.2	• Focus on necessary aspects of technology • Follow tutorials
Time	Not finishing work	0.5	6	3	• Prioritize features • Refer to Gantt chart to stay on schedule
System Fails Testing	System is faulty by deadline	0.3	7	2.1	• Develop a test plan timeline • Conduct tests while developing
Data Loss	Lost Work will delay completion and cost time	0.1	7	0.7	• Have multiple backups • Use source code revision control systems

P: Probability of Occurrence (0-1)
S: Severity of Risk (1-10)
I: Impact of Risk (P * S, 1-10)

6.9: Development Timeline

The development and implementation of this system was completed by dividing the required tasks and assigning them to individuals or groups. In order to ensure the system was completed on time, a Gantt chart was developed to keep group members on task. The Gantt chart can been seen in Figure 6.9.1.



6.9.1: Development Timeline for Whole School Year

7. Conclusions

The MagicTale is now a skeletal backbone of our project objectives and requirements. The implementation of the MagicTale strengthened and developed our skills in Objective-C (object-oriented C programming language) and Sprite kit. However, throughout the course of our project, our team has dealt and overcome many obstacles. We learned that it is important to research various alternative options before designing our project. With requirement changes and design changes, we were forced to cut down unnecessary features and reconsider our alternatives to ensure the completion of our project on time. We realized that our project design required more funding than we expected. Therefore, we had multiple system design changes and reconsidered our project requirements. Our team learned that dealing with system design changes, it is essential to reconsider and prioritize the critical requirements and work closely together. Our team learned that we needed to be flexible and be ready for sudden changes within our project. We also learned that it is important to have alternative solution or backup in the case of any issues that come up. In the near future, we hope to submit the MagicTale onto the IOS app store for users to download and play. We plan to add multiple stage maps, different characters and enemy minions, and additional features.

8. Appendix

```
//
//  MTBoss.h
//  MagicTale
//
//  Created by Lu Cao on 12/25/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTEnemy.h"

@interface MTBoss : MTEnemy

- (id)initAtPosition:(CGPoint)position;

@end
```

```
//
```

```
//  MTBoss.m
//  MagicTale
//
//  Created by Lu Cao on 12/25/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTBoss.h"
#import "MTUKit.h"
@implementation MTBoss

#pragma mark - Initialization
- (id)initAtPosition:(CGPoint)position {

    SKTextureAtlas *atlas = [SKTextureAtlas atlasNamed:@"Slime_Idle"];
    SKTexture *texture = [atlas textureNamed:@"slime_idle_0001"];

    self = [super initWithTexture:texture atPosition:position];
    [self initProperties];

    if (self) {
        self.name = [NSString stringWithFormat:@"Boss"];
        self.attackDamage = 12;
        self.attackRange = 200;
        self.moveSpeed = 50;
        self.alertRange = 300;
        self.attackSpeed = 100;

        // load animation and texture
        [self LoadAnimArrays];
    }

    return self;
}
- (void)LoadAnimArrays {
    self.IdleAnimationFrames = MTLoadFramesFromAtlas(@"Boss_Idle", @"boss_idle_");
    self.WalkAnimationFrames = MTLoadFramesFromAtlas(@"Boss_Walk",
@"boss_walk_");
    self.AttackAnimationFrames = MTLoadFramesFromAtlas(@"Boss_Attack",
@"boss_attack_");
    self.GetHitAnimationFrames = MTLoadFramesFromAtlas(@"Boss_GetHit",
@"boss_getHit_");
    self.DeathAnimationFrames = MTLoadFramesFromAtlas(@"Boss_Death",
@"boss_death_");
}

@end
//
//  MTEnemy.h
//  MagicTale
//
//  Created by Lu Cao on 12/26/13.
```

```
//  Copyright (c) 2013 Lu. All rights reserved.
//

typedef enum : uint8_t {
    MTEnemyTypeMelee = 0,
    MTEnemyTypeRange,
} MTEnemyType;

typedef enum : uint8_t {
    MTEnemyStateIdle = 0,
    MTEnemyStateChase,
    MTEnemyStateAttack,
    MTEnemyStateGetHit,
    MTEnemyStateDeath,
} MTEnemyState;

#import "MTCharacter.h"

@interface MTEnemy : MTCharacter

/* Basic Enemy properties */
@property (nonatomic) MTEnemyType type;                 // melee or range
@property (nonatomic) BOOL foundPlayer;
@property (nonatomic) BOOL attacked;
@property (nonatomic) BOOL gotHit;
@property (nonatomic) CGFloat attackRange;
@property (nonatomic) CGFloat alertRange;

/* Initialize Properties */
- (void)initProperties;
- (void)configurePhysics;

/* Attack, Chase ... */
- (void)startAction:(CGPoint)position withTimeInterval:(NSTimeInterval)interval;
- (void)performAttack;
- (void)applyDamageFromSpell:(MTSpellNode *)spell;

/* update */
- (void)updateWithTimeSinceLastUpdate:(CFTimeInterval)interval;

@end


//
//  MTEnemy.m
//  MagicTale
//
//  Created by Lu Cao on 12/26/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTEnemy.h"
```

```
#import "MTUKit.h"
#import "MTHero.h"

@implementation MTEnemy

#pragma mark - Initialize Properties
- (void)initProperties {
    // properties from current class
    self.requestedAnimation = MTAnimationStateIdle;
    self.type = MTEnemyTypeMelee;
    self.foundPlayer = NO;
    self.attacked = NO;
    self.attackRange = 100;
    self.alertRange = 100;
    self.attackDamage = 1;

    // properties from super class
    self.name = [NSString stringWithFormat:@"Enemy"];

    // configure physical body called in sub classes
//    [self configurePhysics];

}

#pragma mark - AI actions
- (void)startAction:(CGPoint)position withTimeInterval:(NSTimeInterval)interval {
    // if not seen player, stay idle, or move to spawn point and stay idle
    if (!self.foundPlayer && !self.gotHit) {

        if ((MTDistanceBetweenPoints(position, self.position) < self.alertRange)) {
            self.foundPlayer = YES;
        } else if (MTDistanceBetweenPoints(self.position, self.spawnPosition) < 1) {
            self.requestedAnimation = MTAnimationStateIdle;
        } else {
            self.requestedAnimation = MTAnimationStateWalk;
            [self moveTowards:self.spawnPosition withTimeInterval:interval];
        }

    } else {    // seen player, perform action depending on range

        if (MTDistanceBetweenPoints(position, self.position) < self.attackRange * 0.6) {
            self.requestedAnimation = MTAnimationStateAttack;
            [self faceTo:position];
        } else if (MTDistanceBetweenPoints(self.position, position) < self.alertRange * 3) {
            self.requestedAnimation = MTAnimationStateWalk;
            [self moveTowards:position withTimeInterval:interval];
        } else if (self.gotHit) {
            self.requestedAnimation = MTAnimationStateWalk;
            [self moveTowards:position withTimeInterval:interval];
        } else {
            self.foundPlayer = NO;
        }
```

```
    }
}

- (void)performAttack {
    MTHero *player = (MTHero *) [[self parent] childNodeWithName:@"Player"];

    if (self.type == MTEnemyTypeMelee) {
        if (MTDistanceBetweenPoints(self.position, player.position) <= self.attackRange) {
            [player applyDamage:self.attackDamage];
        }
    }

    if (self.type == MTEnemyTypeRange) {
        SKSpriteNode *projectile = [SKSpriteNode
spriteNodeWithImageNamed:@"porcr_projectile"];
        [[self parent] addChild:projectile];
        projectile.position = self.position;
        projectile.zRotation = self.zRotation;
        projectile.name = @"enemyProjectile";
        [projectile setScale:0.215];
        projectile.physicsBody = [SKPhysicsBody
bodyWithRectangleOfSize:projectile.frame.size];
        projectile.physicsBody.categoryBitMask = MTColliderTypeEnemyProjectile;
        projectile.physicsBody.collisionBitMask = 0;
        projectile.physicsBody.contactTestBitMask = MTColliderTypeHero;
        CGFloat rot = self.zRotation;
        [projectile runAction:[SKAction sequence:@[[SKAction moveByX:-sinf(rot)*650
                                                                   y:cosf(rot)*650
                                                            duration:2],
                                                   [SKAction removeFromParent]]]];
    }
}

// if attack by player, found player
- (void)applyDamageFromSpell:(MTSpellNode *)spell {
    [super applyDamageFromSpell:spell];
    [self runAction:[SKAction sequence:@[[SKAction runBlock:^{self.gotHit = YES;}],
                                         [SKAction waitForDuration:5],
                                         [SKAction runBlock:^{self.gotHit = NO;}]]]];
}

#pragma mark - Configure physics body and collision
- (void)configurePhysics {
    self.physicsBody = [SKPhysicsBody bodyWithCircleOfRadius:self.bodyRadius];

    // Our object type for collisions.
    self.physicsBody.categoryBitMask = MTColliderTypeEnemy;

    self.physicsBody.collisionBitMask = MTColliderTypeWall | MTColliderTypeEnemy |
                                        MTColliderTypeWater | MTColliderTypeTrap |
                                        MTColliderTypeRock | MTColliderTypeTree |
```

```
                                MTColliderTypeWater | MTColliderTypeFence |
MTColliderTypeHouse;
    // We want notifications for colliding with these objects.
    self.physicsBody.contactTestBitMask = MTColliderTypeHero | MTColliderTypeSpell |
MTColliderTypeTrap;

}

- (void)updateWithTimeSinceLastUpdate:(CFTimeInterval)interval {
    [super updateWithTimeSinceLastUpdate:interval];
    if (self.hp > 0) {
        CGPoint playerPosotion = [[self parent] childNodeWithName:@"Player"].position;
        [self startAction:playerPosotion withTimeInterval:interval];
    }
}

@end
```

```
//
//  MTMinion.h
//  MagicTale
//
//  Created by Lu Cao on 1/9/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTEnemy.h"

@interface MTMinion : MTEnemy

- (id)initWithName:(NSString *)name atPosition:(CGPoint)position;

@end
```

```
//
//  MTMinion.m
//  MagicTale
//
//  Created by Lu Cao on 1/9/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTMinion.h"
```

```
@implementation MTMinion

- (id)initWithName:(NSString *)name atPosition:(CGPoint)position {

    // working with strings
    NSString *lowercaseName;
    NSString *uppercaseName;
    lowercaseName = [name stringByReplacingCharactersInRange:NSMakeRange(0,1)
                                        withString:[[name substringToIndex:1] lowercaseString]];
    uppercaseName = [name stringByReplacingCharactersInRange:NSMakeRange(0,1)
                                        withString:[[name substringToIndex:1] capitalizedString]];
    NSLog(@"%@",lowercaseName);
    NSLog(@"%@",uppercaseName);

    NSString *idleAtlas = [uppercaseName stringByAppendingString:@"_Idle"];
    NSString *walkAtlas = [uppercaseName stringByAppendingString:@"_Walk"];
    NSString *attackAtlas = [uppercaseName stringByAppendingString:@"_Attack"];
    NSString *getHitAtlas = [uppercaseName stringByAppendingString:@"_GetHit"];
    NSString *deathAtlas = [uppercaseName stringByAppendingString:@"_Death"];
    NSString *idleFrames = [lowercaseName stringByAppendingString:@"_idle_"];
    NSString *walkFrames = [lowercaseName stringByAppendingString:@"_walk_"];
    NSString *attackFrames = [lowercaseName stringByAppendingString:@"_attack_"];
    NSString *getHitFrames = [lowercaseName stringByAppendingString:@"_getHit_"];
    NSString *deathFrames = [lowercaseName stringByAppendingString:@"_death_"];

    SKTextureAtlas *atlas = [SKTextureAtlas atlasNamed:idleAtlas];
    SKTexture *texture = [atlas textureNamed:@""];

    self = [super initWithTexture:texture atPosition:position];
    self.texture = [SKTexture textureWithImageNamed:[idleFrames
stringByAppendingString:@"0001"]];

    // load textures
    self.IdleAnimationFrames = MTLoadFramesFromAtlas(idleAtlas, idleFrames);
    self.WalkAnimationFrames = MTLoadFramesFromAtlas(walkAtlas, walkFrames);
    self.AttackAnimationFrames = MTLoadFramesFromAtlas(attackAtlas, attackFrames);
    self.GetHitAnimationFrames = MTLoadFramesFromAtlas(getHitAtlas, getHitFrames);
    self.DeathAnimationFrames = MTLoadFramesFromAtlas(deathAtlas, deathFrames);

    // read from plist and set up attributes
    self.path = [[NSBundle mainBundle] pathForResource:@"enemy" ofType:@"plist"];
    self.dict = [[NSMutableDictionary alloc] initWithContentsOfFile:self.path];

    NSDictionary *dictAttributes = [[self.dict objectForKey:name]
objectForKey:@"Attributes"];

    NSString *attackType = [dictAttributes valueForKey:@"AttackType"];
    if ([attackType isEqualToString:@"Melee"])
        self.type = MTEnemyTypeMelee;
    if ([attackType isEqualToString:@"Range"])
        self.type = MTEnemyTypeRange;
```

```
    self.maxHP = [[dictAttributes valueForKey:@"MaxHP"] floatValue];
    self.maxMP = [[dictAttributes valueForKey:@"MaxMP"] floatValue];
    self.attackSpeed = [[dictAttributes valueForKey:@"AttackSpeed"] floatValue];
    self.moveSpeed = [[dictAttributes valueForKey:@"MoveSpeed"] floatValue];
    self.hitRecoverySpeed = [[dictAttributes valueForKey:@"HitRecoverySpeed"]
floatValue];
    self.attackDamage = [[dictAttributes valueForKey:@"AttackDamage"] floatValue];
    self.bodyRadius = [[dictAttributes valueForKey:@"BodyRadius"] floatValue];
    self.alertRange = [[dictAttributes valueForKey:@"AlertRange"] floatValue];
    self.attackRange = [[dictAttributes valueForKey:@"AttackRange"] floatValue];

    // setups
    self.hp = self.maxHP;
    self.mp = self.maxMP;
    self.requestedAnimation = MTAnimationStateIdle;

    [super configurePhysics];

    // debug draw
    SKShapeNode *debugCircle = [[SKShapeNode alloc] init];
    CGMutablePathRef debugCirclePath = CGPathCreateMutable();
    CGPathAddArc(debugCirclePath, NULL, 0, 0, self.bodyRadius, 0, M_PI*2, NO);
    debugCircle.path = debugCirclePath;
    debugCircle.strokeColor = [SKColor redColor];
    [self addChild:debugCircle];

    // attack range draw
    SKShapeNode *attackRangeCircle = [[SKShapeNode alloc] init];
    CGMutablePathRef attackRangeCirclePath = CGPathCreateMutable();
    CGPathAddArc(attackRangeCirclePath, NULL, 0, 0, self.attackRange*2, 0, M_PI*2,
NO);
    attackRangeCircle.path = attackRangeCirclePath;
    attackRangeCircle.strokeColor = [SKColor greenColor];
    [self addChild:attackRangeCircle];

    return self;
}

@end



//
//  MTCharacter.h
//  MagicTale
//
//  Created by Lu Cao on 12/24/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

/* The different animation states of an animated character. */
```

```
typedef enum : uint8_t {
    MTAnimationStateIdle = 0,
    MTAnimationStateWalk,
    MTAnimationStateAttack,
    MTAnimationStatePreAttack,
    MTAnimationStateHoldAttack,
    MTAnimationStatePostAttack,
    MTAnimationStateGetHit,
    MTAnimationStateDeath,
    kAnimationStateCount
} MTAnimationState;

/* Define current status of character (burn, freeze, poisoned ...) */
typedef enum : uint8_t {
    MTAbnormalStatusBurning = 0,
    MTAbnormalStatusChilled = 1,
    MTAbnormalStatusSlowed = 2,
    MTAbnormalStatusFreezing = 3,
    MTAbnormalStatusStunned = 4,
} MTAbnormalStatus;

/* Bitmask for the different entities with physics bodies. */
typedef enum : u_int32_t {
    MTColliderTypeHero           = 0x1 << 0,
    MTColliderTypeEnemy          = 0x1 << 1,
    MTColliderTypeEnemyProjectile  = 0x1 << 2,
    MTColliderTypeSpell          = 0x1 << 3,
    MTColliderTypeWall           = 0x1 << 4,
    MTColliderTypeTrap           = 0x1 << 5,
    MTColliderTypeWater          = 0x1 << 6,
    MTColliderTypeRock           = 0x1 << 7,
    MTColliderTypeTree           = 0x1 << 8,
    MTColliderTypeHouse          = 0x1 << 9,
    MTColliderTypeFence          = 0x1 << 10
} MTColliderType;

#import <SpriteKit/SpriteKit.h>
#import "MTSpellNode.h"

@class MTSpellNode;
@interface MTCharacter : SKSpriteNode

/* Preload shared animation frames, emitters, etc. */
+ (void)loadSharedAssets;

/* Basic attributes */
@property (nonatomic) CGFloat hp;
@property (nonatomic) CGFloat maxHP;
@property (nonatomic) CGFloat mp;
@property (nonatomic) CGFloat maxMP;
@property (nonatomic) SKSpriteNode *overheadHPBar;
@property (nonatomic) CGFloat moveSpeed;
```

```
@property (nonatomic) CGFloat attackDamage;
@property (nonatomic) CGFloat attackSpeed;
@property (nonatomic) CGFloat hitRecoverySpeed;
@property (nonatomic) CGFloat bodyRadius;

@property (nonatomic) CGPoint spawnPosition;
@property (nonatomic) CGPoint userTouchedPosition;

/* plist bundle (init in subclasses) */
@property (nonatomic) NSString *path;
@property (nonatomic) NSMutableDictionary *dict;

/* Current Abnormal Status */
@property (nonatomic) MTAbnormalStatus *currentAbnormalStatus;

/* Basic Animation Arrays that generally fit */
@property (nonatomic) MTAnimationState requestedAnimation;
@property (nonatomic) NSString *activeAnimationKey;
@property (nonatomic) NSArray *idleAnimationFrames;
@property (nonatomic) NSArray *walkAnimationFrames;
@property (nonatomic) NSArray *attackAnimationFrames;          // general attack frames,
undivided
@property (nonatomic) NSArray *preAttackAnimationFrames;       // pre, hold, post for
hero
@property (nonatomic) NSArray *holdAttackAnimationFrames;
@property (nonatomic) NSArray *postAttackAnimationFrames;
@property (nonatomic) NSArray *getHitAnimationFrames;
@property (nonatomic) NSArray *deathAnimationFrames;

/* Booleans indicator current state */
@property (nonatomic) BOOL isDead;
@property (nonatomic) BOOL canAttack;
@property (nonatomic) BOOL isAttacking;
@property (nonatomic) BOOL canMove;
@property (nonatomic) BOOL isGettingHit;           // for hit recovery, can't do nothing
when get hit

/* Resolve requested animation */
- (void)resolveRequestedAnimation;
- (void)fireAnimationForState:(MTAnimationState)animationState
usingTextures:(NSArray *)frames withKey:(NSString *)key;

/* Resolve abnormal animation */
- (void)resolveAbnormalStatus:(MTAbnormalStatus)status withDamge:(CGFloat)damage
andDuration:(CGFloat)duration;

/* Initialize a standard sprite. */
- (id)initWithTexture:(SKTexture *)texture atPosition:(CGPoint)position;

- (void)setupOverheadHPBar;

/* Configure physical body */
```

```
- (void)collidedWith:(SKSpriteNode *)other;

/* Orientation, Movement, and Attacking. */
- (void)moveTowards:(CGPoint)position withTimeInterval:(NSTimeInterval)timeInterval;
- (void)faceTo:(CGPoint)position;
- (void)performAttack;                                  // dealt damage
- (void)applyDamage:(CGFloat)damage;                    // receive damage
- (void)applyDamageFromSpell:(MTSpellNode *)spell;      // receive damage from
a spell

/* Loop update */
- (void)updateWithTimeSinceLastUpdate:(CFTimeInterval)interval;

@end
//
//  MTCharacter.m
//  MagicTale
//
//  Created by Lu Cao on 12/24/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTCharacter.h"
#import "MTSpellNode.h"
#import "MTFloatingDamage.h"
#import "MTUKit.h"

@implementation MTCharacter

#pragma mark - Initialization
- (id)initWithTexture:(SKTexture *)texture atPosition:(CGPoint)position {
    self = [super initWithTexture:texture];

    if (self) {
        self.position = position;
        self.spawnPosition = position;
        [self initAttributes];
        [self configurePhysics];

        // debug draw, deactive by comment it
//        [self drawDebug];
    }

    return self;
}

// called after initialization
- (void)setupOverheadHPBar {
    self.overheadHPBar = [SKSpriteNode
spriteNodeWithImageNamed:@"overheadHPBar"];
    self.overheadHPBar.size = CGSizeMake(self.size.width/3, self.size.height/50);
    [[self parent] addChild:self.overheadHPBar];
```

```
}

#pragma mark - Init Attributes and Physics
- (void)initAttributes {
    self.hp = 100;
    self.maxHP = 100;
    self.mp = 100;
    self.maxMP = 100;
    self.attackDamage = 1;
    self.attackSpeed = 100;
    self.moveSpeed = 100;
    self.hitRecoverySpeed = 100;
    self.bodyRadius = 25;

    self.isDead = NO;
    self.canAttack = NO;
    self.isAttacking = NO;
    self.canMove = YES;
    self.isGettingHit = NO;
}

- (void)configurePhysics {
    // overwritten by subclasses
}

- (void)collidedWith:(SKPhysicsBody *)other {
    // Handle a collision with another character, projectile, wall, etc (usually overidden).
}

#pragma mark - Animation
- (void)resolveRequestedAnimation {
    // Determine the animation we want to play.
    NSString *animationKey = nil;
    NSArray *animationFrames = nil;
    MTAnimationState animationState = self.requestedAnimation;

    switch (animationState) {

        default:
        case MTAnimationStateIdle:
            animationKey = @"anim_idle";
            animationFrames = self.idleAnimationFrames;
            break;

        case MTAnimationStateWalk:
            animationKey = @"anim_walk";
            animationFrames = self.walkAnimationFrames;
            break;

        case MTAnimationStateAttack:
            animationKey = @"anim_attack";
            animationFrames = self.attackAnimationFrames;
```

```
            break;

        case MTAnimationStateGetHit:
            animationKey = @"anim_getHit";
            animationFrames = self.getHitAnimationFrames;
            break;

        case MTAnimationStateDeath:
            animationKey = @"anim_death";
            animationFrames = self.deathAnimationFrames;
            break;

    }

    if (animationKey)
        [self fireAnimationForState:animationState usingTextures:animationFrames
withKey:animationKey];
}

- (void)fireAnimationForState:(MTAnimationState)animationState
usingTextures:(NSArray *)frames withKey:(NSString *)key {

    SKAction *animAction = [self actionForKey:key];
    if ([frames count] < 1) {
        return;                                 // we already have a running animation or there
aren't any frames to animate
    }
    if (animAction && self.requestedAnimation != MTAnimationStateGetHit) {
        return;
    }

    self.activeAnimationKey = key;

    if ([key isEqualToString:@"anim_attack"] && self.canAttack && !self.isGettingHit) {
        self.canMove = NO;
        self.canAttack = NO;
        [self faceTo:self.userTouchedPosition];
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.attackSpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_walk"] && self.canMove) {
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.moveSpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_getHit"]) {
        self.isGettingHit = YES;
        self.canAttack = NO;
        self.canMove = NO;
```

```
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.hitRecoverySpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_death"]) {
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:1.0f/frames.count resize:YES restore:NO],
                                             [SKAction waitForDuration:3],
                                             [SKAction fadeAlphaTo:0 duration:3],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_idle"]) {
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:1.0f/frames.count resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
         withKey:key];
    }
}

- (void)animationHasCompleted:(MTAnimationState)animationState {
    switch (animationState) {
        default:
            break;

        case MTAnimationStateAttack:
            if (self.isAttacking)
                self.isAttacking = NO;
            if (!self.canMove)
                self.canMove = YES;
            if (!self.canAttack)
                self.canAttack = YES;
            [self performAttack];
            break;

        case MTAnimationStateGetHit:
            if (self.isGettingHit)
                self.isGettingHit = NO;
            if (!self.canMove)
                self.canMove = YES;
            if (!self.canAttack)
                self.canAttack = YES;
            break;

        case MTAnimationStateDeath:
            [self removeFromParent];
            break;
    }
    self.activeAnimationKey = nil;
}
```

```
#pragma mark - Movement
- (void)moveTowards:(CGPoint)position withTimeInterval:(NSTimeInterval)timeInterval {
    // do nothing if dead
    if (self.isDead)
        return;
    // can't do nothing when in hit recovery
    if (self.isGettingHit)
        return;

    // can move state decide whether can or not move
    if (!self.canMove)
        return;

    CGPoint curPosition = self.position;
    CGFloat dx = position.x - curPosition.x;
    CGFloat dy = position.y - curPosition.y;
    CGFloat dt = self.moveSpeed * timeInterval;

    CGFloat ang = MT_POLAR_ADJUST(MTRadiansBetweenPoints(position,
curPosition));
    self.zRotation = ang;

    CGFloat distRemaining = hypotf(dx, dy);
    if (distRemaining < dt) {
        self.position = position;
    } else {
        self.position = CGPointMake(curPosition.x - sinf(ang)*dt,
                                    curPosition.y + cosf(ang)*dt);
    }
}

- (void)faceTo:(CGPoint)position {
    // do nothing if dead
    if (self.isDead)
        return;

    // can't do nothing when in hit recovery
    if (self.isGettingHit)
        return;

    CGFloat ang = MT_POLAR_ADJUST(MTRadiansBetweenPoints(position,
self.position));
    SKAction *action = [SKAction rotateToAngle:ang duration:0];
    [self runAction:action];
    self.zRotation = ang;
}

#pragma mark - attack
// attack type (melee, range, magic) and dealt damage
- (void)performAttack {
    // overwritten by subclasses
```

```
}

- (void)applyDamage:(CGFloat)damage
{
    if (damage <= 0)
        return;

    self.hp -= damage;

    // floating damage counter
    MTFloatingDamage *counter = [[MTFloatingDamage alloc] initAtPosition:self.position
ofSprite:self withDamage:damage];
    [[self parent] addChild:counter];
    [counter anim];

    if (self.hp <= 0) {
        self.isDead = YES;
        [self.overheadHPBar removeFromParent];
        return;
    }
    self.requestedAnimation = MTAnimationStateGetHit;
    self.isGettingHit = YES;
}

- (void)applyDamageFromSpell:(MTSpellNode *)spell {
    [self applyDamage:spell.damage];
    [self resolveAbnormalStatus:(MTAbnormalStatus)spell.status
withDamge:spell.statusDamage andDuration:spell.statusDuration];
}

#pragma mark - abnormal status
- (void)resolveAbnormalStatus:(MTAbnormalStatus)status withDamge:(CGFloat)damage
andDuration:(CGFloat)duration {
    SKAction *changeColor = [SKAction runBlock:^{self.colorBlendFactor = 0.5;}];
    SKAction *changeColorBack = [SKAction runBlock:^{self.colorBlendFactor = 0;}];

    switch (status) {
        default:
            break;

        case MTAbnormalStatusChilled: {
            self.color = [SKColor blueColor];
            [self runAction:[SKAction sequence:@[changeColor,
                                                 [SKAction runBlock:^{self.moveSpeed *= 0.3;}],
                                                 [SKAction waitForDuration:duration],
                                                 [SKAction runBlock:^{self.moveSpeed /= 0.3;}],
                                                 changeColorBack]]];
        }
            break;

        case MTAbnormalStatusBurning: {
            self.color = [SKColor redColor];
```

```
        SKAction *repeat = [SKAction repeatAction:[SKAction sequence:@[[SKAction
waitForDuration:2],
                                                                       [SKAction runBlock:^{[self
applyDamage:damage];}]]]
                                    count:4];
        SKAction *seq = [SKAction sequence:@[changeColor,
                                             repeat,
                                             [SKAction waitForDuration:1],
                                             changeColorBack]];
        [self runAction:seq];
    }
        break;


  }
}

#pragma mark - Loop Update
- (void)updateWithTimeSinceLastUpdate:(CFTimeInterval)interval {

  // sync overhead hp bar
  self.overheadHPBar.position = CGPointMake(self.position.x, self.position.y +
self.size.height/1.8);
  [self.overheadHPBar setXScale:self.hp/self.maxHP];

  if (self.isDead) {
    self.requestedAnimation = MTAnimationStateDeath;
  }

  if (!self.isAttacking)
    self.canAttack = YES;

  [self resolveRequestedAnimation];
}

#pragma mark - Draw Debug
- (void)drawDebug {
  SKShapeNode *circle = [[SKShapeNode alloc] init];
  CGMutablePathRef circlePath = CGPathCreateMutable();
  CGPathAddArc(circlePath, NULL, 0, 0, self.bodyRadius, 0, M_PI*2, NO);
  circle.path = circlePath;
  circle.strokeColor = [SKColor redColor];
  [self addChild:circle];

}

#pragma mark - Shared Assets
+ (void)loadSharedAssets {
  // overridden by subclasses
}

@end
```

```
//
//  MTYoungWitch.h
//  MagicTale
//
//  Created by Lu Cao on 12/24/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTCharacter.h"

@interface MTHero : MTCharacter

@property (nonatomic) NSString *pPath;            // path of plist stored in sandbox
@property (nonatomic) NSMutableDictionary *pDict;   // sandbox dict

// spell node
@property (nonatomic) MTSpellType selectedSpellType;
@property (nonatomic) MTSpellNode *currentReadySpell;
- (void)holdSpell;  // generate spell at hero's position, make it ready to cast out

// exp
@property (nonatomic) CGFloat exp;
@property (nonatomic) CGFloat levelUpExp;
@property (nonatomic) NSInteger level;
- (void)levelUp;

// player's current move state
@property (nonatomic) BOOL canInteract;
@property (nonatomic) NSInteger castingStates;
@property (nonatomic) BOOL isAiming;
@property (nonatomic) BOOL isHoldingSpell;
@property (nonatomic) BOOL moveRequested;
@property (nonatomic) BOOL castRequested;

- (id)initAtPosition:(CGPoint)position;
- (void)configurePhysics;

@end
```

```
//
//  MTYoungWitch.m
//  MagicTale
//
//  Created by Lu Cao on 12/24/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//
```

```
#import "MTHero.h"
#import "MTUKit.h"

@implementation MTHero

#pragma mark - Initialization
- (id)initAtPosition:(CGPoint)position {

    SKTextureAtlas *atlas = [SKTextureAtlas atlasNamed:@"Hero_Idle"];
    SKTexture *texture = [atlas textureNamed:@"hero_idle_0001"];

    self = [super initWithTexture:texture atPosition:position];

    if (self) {
        // setup attributes
        [self setupAttributes];

        // load animation and texture
        [self LoadAnimArrays];

        // configure physical body
        [self configurePhysics];

        [self initMisc];
    }

    return self;
}

- (void)initMisc {
    self.name = [NSString stringWithFormat:@"Player"];
    self.userTouchedPosition = self.position;

    self.canInteract = YES;
    self.castingStates = 0;
    self.isAiming = NO;
    self.isHoldingSpell = NO;
    self.moveRequested = NO;
    self.castRequested = NO;
}

- (void)setupAttributes {
    // init plist
    self.path = [[NSBundle mainBundle] pathForResource:@"player" ofType:@"plist"];
    self.dict = [[NSMutableDictionary alloc] initWithContentsOfFile:self.path];
    self.pPath = [[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) objectAtIndex:0]
                  stringByAppendingString:@"/player.plist"];

    // copy plist to sandbox if not exist
    if (![[NSFileManager defaultManager] fileExistsAtPath:self.pPath])
```

```
        [[NSFileManager defaultManager] copyItemAtPath:self.path toPath:self.pPath
error:nil];
    self.pDict = [[NSMutableDictionary alloc] initWithContentsOfFile:self.pPath];

    // read from plist (currently from editor, not from user path) dict, pdict
    NSDictionary *dictAttributes = [self.dict objectForKey:@"Attributes"];
    NSLog(@"Player Attributes \n %@", dictAttributes);

    self.level            = [[dictAttributes valueForKey:@"Level"] integerValue];
    self.levelUpExp       = [[dictAttributes valueForKey:@"NextLevelExp"] floatValue];
    self.maxHP            = [[dictAttributes valueForKey:@"MaxHP"] floatValue];
    self.maxMP            = [[dictAttributes valueForKey:@"MaxMP"] floatValue];
    self.attackSpeed      = [[dictAttributes valueForKey:@"AttackSpeed"] floatValue];
    self.moveSpeed        = [[dictAttributes valueForKey:@"MoveSpeed"] floatValue];
    self.hitRecoverySpeed = [[dictAttributes valueForKey:@"HitRecoverySpeed"]
floatValue];
    self.attackDamage     = [[dictAttributes valueForKey:@"AttackDamage"] floatValue];
    self.bodyRadius       = [[dictAttributes valueForKey:@"BodyRadius"] floatValue];


    // setups
    self.hp = self.maxHP;
    self.mp = self.maxMP;
    self.exp = 0;

}


- (void)LoadAnimArrays {
    self.idleAnimationFrames       = MTLoadFramesFromAtlas(@"Hero_Idle",
@"hero_idle_");
    self.walkAnimationFrames       = MTLoadFramesFromAtlas(@"Hero_Walk",
@"hero_walk_");
    self.preAttackAnimationFrames  = MTLoadFramesFromAtlas(@"Hero_PreAttack",
@"hero_preAttack_");
    self.holdAttackAnimationFrames = MTLoadFramesFromAtlas(@"Hero_HoldAttack",
@"hero_holdAttack_");
    self.postAttackAnimationFrames = MTLoadFramesFromAtlas(@"Hero_PostAttack",
@"hero_postAttack_");
    self.getHitAnimationFrames     = MTLoadFramesFromAtlas(@"Hero_GetHit",
@"hero_getHit_");
    self.deathAnimationFrames      = MTLoadFramesFromAtlas(@"Hero_Death",
@"hero_death_");
}

#pragma mark - configure hero animation (post attack etc.)
- (void)resolveRequestedAnimation {
    // Determine the animation we want to play.
    NSString *animationKey = nil;
    NSArray *animationFrames = nil;
    MTAnimationState animationState = self.requestedAnimation;
```

```
    switch (animationState) {

        default:
        case MTAnimationStateIdle:
            animationKey = @"anim_idle";
            animationFrames = self.idleAnimationFrames;
            break;

        case MTAnimationStateWalk:
            animationKey = @"anim_walk";
            animationFrames = self.walkAnimationFrames;
            break;

        case MTAnimationStatePreAttack:
            animationKey = @"anim_preAttack";
            animationFrames = self.preAttackAnimationFrames;
            break;

        case MTAnimationStateHoldAttack:
            animationKey = @"anim_holdAttack";
            animationFrames = self.holdAttackAnimationFrames;
            break;

        case MTAnimationStatePostAttack:
            animationKey = @"anim_postAttack";
            animationFrames = self.postAttackAnimationFrames;
            break;

        case MTAnimationStateGetHit:
            animationKey = @"anim_getHit";
            animationFrames = self.getHitAnimationFrames;
            break;

        case MTAnimationStateDeath:
            animationKey = @"anim_death";
            animationFrames = self.deathAnimationFrames;
            break;

    }

    if (animationKey)
        [self fireAnimationForState:animationState usingTextures:animationFrames
withKey:animationKey];
}

- (void)fireAnimationForState:(MTAnimationState)animationState
usingTextures:(NSArray *)frames withKey:(NSString *)key {

    SKAction *animAction = [self actionForKey:key];
    if ([frames count] < 1) {
        return;                                 // we already have a running animation or there
aren't any frames to animate
```

```
    }
    if (animAction) {
        return;
    }

    self.activeAnimationKey = key;

    if ([key isEqualToString:@"anim_preAttack"]) {
//        NSLog(@"PLAYER ATTACK");
        self.castingStates = 1;
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.attackSpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_holdAttack"]) {
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.attackSpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_postAttack"]) {
        [self performAttack];
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.attackSpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_walk"] && self.canMove) {
//        NSLog(@"PLAYER WALK");
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.moveSpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_getHit"]) {
//        NSLog(@"PLAYER GOT HIT");
        self.canMove = false;
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:(1.0f/frames.count)*(100/self.hitRecoverySpeed) resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    } else if ([key isEqualToString:@"anim_death"]) {
//        NSLog(@"PLAYER DEAD");
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:1.0f/frames.count resize:YES restore:NO],
                                             [SKAction waitForDuration:3],
                                             [SKAction fadeAlphaTo:0 duration:3],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
```

```
    } else if ([key isEqualToString:@"anim_idle"]) {
//        NSLog(@"PLAYER IDLE");
        [self runAction:[SKAction sequence:@[[SKAction animateWithTextures:frames
timePerFrame:1.0f/frames.count resize:YES restore:NO],
                                             [SKAction runBlock:^{[self
animationHasCompleted:animationState];}]]]
                withKey:key];
    }
}

- (void)animationHasCompleted:(MTAnimationState)animationState {
    switch (animationState) {
        default:
            break;

        case MTAnimationStatePreAttack:
            [self holdSpell];
            if (self.castingStates == 2) {
                self.requestedAnimation = MTAnimationStateHoldAttack;
            } else {
                self.castingStates = 0;
                self.requestedAnimation = MTAnimationStatePostAttack;
            }
            break;

        case MTAnimationStateHoldAttack:

            break;

        case MTAnimationStatePostAttack:
            self.requestedAnimation = MTAnimationStateIdle;
            self.canInteract = YES;
            self.castingStates = 0;
            break;

        case MTAnimationStateGetHit:
            if (self.isGettingHit)
                self.isGettingHit = NO;
            if (!self.canMove)
                self.canMove = YES;
            if (!self.canAttack)
                self.canAttack = YES;
            self.requestedAnimation = MTAnimationStateIdle;
            break;

        case MTAnimationStateDeath:
            [self removeFromParent];
            break;
    }
    self.activeAnimationKey = nil;
}
```

```
#pragma mark - configure physics
- (void)configurePhysics {

    self.physicsBody = [SKPhysicsBody bodyWithCircleOfRadius:self.bodyRadius];

    // Our object type for collisions.
    self.physicsBody.categoryBitMask = MTColliderTypeHero;

    // pretty much will collides with everything except own casted magic
    self.physicsBody.collisionBitMask = MTColliderTypeWall | MTColliderTypeEnemy |
                                        MTColliderTypeWater | MTColliderTypeTrap |
                                        MTColliderTypeRock | MTColliderTypeTree |
                                        MTColliderTypeWater | MTColliderTypeFence |
MTColliderTypeHouse;

    // We want notifications for colliding with these objects.
    self.physicsBody.contactTestBitMask = MTColliderTypeEnemy | MTColliderTypeTrap
| MTColliderTypeEnemyProjectile;
}

#pragma mark - ready and perform attack
// hold ready spell on top of hero
- (void)holdSpell {

    self.currentReadySpell = [[MTSpellNode alloc] initAtPosition:self.position
                                             withSpellType:self.selectedSpellType
                                          andDamageModifier:self.attackDamage];
    self.currentReadySpell.spellType = self.selectedSpellType;
    [self.currentReadySpell readySpellWithPosition:self.position];
    [[self parent] addChild:self.currentReadySpell];

}

// cast it out
- (void)performAttack {

    /*
    // generate an empty spell if not enough mp
    if (self.mp < self.currentReadySpell.cost) {
        self.selectedSpellType = MTSpellTypeNoMana;
        self.currentReadySpell = [[MTSpellNode alloc] initAtPosition:self.position
                                                 withSpellType:self.selectedSpellType
                                              andDamageModifier:self.attackDamage];
        [[self parent] addChild:self.currentReadySpell];
        self.currentReadySpell.position = self.position;
        [self.currentReadySpell castSpellWithPosition:self.userTouchedPosition];

        self.requestedAnimation = MTAnimationStateIdle;

        return;
    }
     */
```

```
    self.currentReadySpell.position = self.position;
    [self.currentReadySpell castSpellWithPosition:self.userTouchedPosition];

    self.requestedAnimation = MTAnimationStatePostAttack;

    self.mp -= self.currentReadySpell.cost;
}

-(void)levelUp {
    if (self.exp >= self.levelUpExp) {
        NSLog(@"Player Level UP!");
        self.exp = 0;

        // write new attributes into sandbox player plist
        // something like max hp...
        [self.pDict writeToFile:self.pPath atomically:YES];
    }
}

- (void)updateWithTimeSinceLastUpdate:(CFTimeInterval)interval {
    [super updateWithTimeSinceLastUpdate:interval];

    // restore hp and mp
    if (self.hp <= self.maxHP) {
        if (self.hp >= self.maxHP*0.99999)
            self.hp = self.maxHP;
        else self.hp += self.maxHP*0.00001;
    }
    if (self.mp <= self.maxMP) {
        if (self.mp >= self.maxMP*0.9999)
            self.mp = self.maxMP;
        else self.mp += self.maxMP*0.0001;
    }
}

+ (void)loadSharedAssets {

}

@end



//
//  MTSetting.h
//  magictale
//
//  Created by Yetian Mao on 5/6/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//
```

```
#import <SpriteKit/SpriteKit.h>

@interface MTSetting : SKScene
@property (nonatomic) SKSpriteNode *onButton;
@property (nonatomic) SKSpriteNode *offButton;
@property (nonatomic) SKSpriteNode *chinese;
@property (nonatomic) SKSpriteNode *japanese;
@property (nonatomic) SKSpriteNode *english;
- (id)initWithSize:(CGSize)size;
@end



//
//  MTSetting.m
//  magictale
//
//  Created by Yetian Mao on 5/6/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTSetting.h"
#import "MTStartUpScene.h"

@implementation MTSetting


- (id)initWithSize:(CGSize)size {
    if (self = [super initWithSize:size]) {
        SKSpriteNode *bg = [SKSpriteNode spriteNodeWithImageNamed:@"Background"];
        bg.position = CGPointMake(self.size.width / 2, self.size.height / 2);
        [bg setScale:0.227];
        SKSpriteNode *soundButton = [SKSpriteNode
spriteNodeWithImageNamed:@"sound1"];
        soundButton.position = CGPointMake(80, self.size.height/2 + 50);
        soundButton.name = @"sound";
        [soundButton setScale:0.3];

        _onButton = [SKSpriteNode spriteNodeWithImageNamed:@"on2"];
        _onButton.position = CGPointMake(200, self.size.height/2 + 50);
        _onButton.name = @"on";
        [_onButton setScale:0.3];
        _offButton = [SKSpriteNode spriteNodeWithImageNamed:@"off1"];
        _offButton.position = CGPointMake(300, self.size.height/2 + 50);
        _offButton.name = @"off";
        [_offButton setScale:0.3];

        SKSpriteNode *languageButton = [SKSpriteNode
spriteNodeWithImageNamed:@"language1"];
        languageButton.position = CGPointMake(80, self.size.height/2 - 25);
        languageButton.name = @"language1";
        [languageButton setScale:0.3];
```

```
        _chinese = [SKSpriteNode spriteNodeWithImageNamed:@"chinese1"];
        _chinese.position = CGPointMake(235, self.size.height/2 - 25);
        _chinese.name = @"chinese";
        [_chinese setScale:0.3];
        _japanese = [SKSpriteNode spriteNodeWithImageNamed:@"japanese1"];
        _japanese.position = CGPointMake(390, self.size.height/2 - 25);
        _japanese.name = @"japanese";
        [_japanese setScale:0.3];
        _english = [SKSpriteNode spriteNodeWithImageNamed:@"english2"];
        _english.position = CGPointMake(235, self.size.height/2 - 100);
        _english.name = @"english";
        [_english setScale:0.3];

        SKSpriteNode *returnbutton = [SKSpriteNode
spriteNodeWithImageNamed:@"returnbutton"];
        returnbutton.position = CGPointMake(30, 280);
        returnbutton.name = @"BackButton";
        [returnbutton setScale:0.2];

        [self addChild:bg];
        [self addChild:soundButton];
        [self addChild:_onButton];
        [self addChild:_offButton];
        [self addChild:languageButton];
        [self addChild:_chinese];
        [self addChild:_japanese];
        [self addChild:_english];
        [self addChild:returnbutton];
    }

    return self;
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    /* Called when a touch begins */

    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInNode:self];
    SKNode *node = [self nodeAtPoint:location];
    SKTransition *reveal = [SKTransition fadeWithDuration:1];
    //NSLog(@"%@", node.name);
    if ([node.name isEqualToString:@"on"]) {

    }
    if ([node.name isEqualToString:@"off"]) {

    }
    if ([node.name isEqualToString:@"BackButton"]) {
        MTStartUpScene *scene1= [MTStartUpScene
sceneWithSize:self.view.bounds.size];
        scene1.scaleMode = SKSceneScaleModeResizeFill;
```

```
        [self.view presentScene:scene1 transition:reveal];
    }

}


@end


//
//  MTCutScene.h
//  magictale
//
//  Created by Yetian Mao on 5/6/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>

@interface MTCutScene : SKScene
- (id)initWithSize:(CGSize)size;
@end



//
//  MTCutScene.m
//  magictale
//
//  Created by Yetian Mao on 5/6/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTCutScene.h"
#import "MTStartUpScene.h"
#import "MTStageSelectScene.h"

@implementation MTCutScene{
    NSString *content1;
    NSInteger i, counter, changeto;
    NSString *path;
    SKSpriteNode *pork;
    SKLabelNode *porktalk;
    SKSpriteNode *magicbook;
    NSMutableArray *conversation;
}

- (id)initWithSize:(CGSize)size {
    if (self = [super initWithSize:size]) {
        path = [[NSBundle mainBundle] pathForResource:@"conversation" ofType:@"plist"];
        conversation = [[NSMutableArray alloc] initWithContentsOfFile:path];
        counter = [conversation count];
```

```
        content1 = @"...";
        changeto = 1;

        SKSpriteNode *bg = [SKSpriteNode
spriteNodeWithImageNamed:@"greybackground"];
        bg.position = CGPointMake(self.size.width / 2, self.size.height / 2);
        [bg setScale:0.227];
        pork = [SKSpriteNode spriteNodeWithImageNamed:@"porkicon"];
        pork.position = CGPointMake(self.size.width + pork.size.width/2, 80);
        porktalk = [SKLabelNode labelNodeWithFontNamed:@"Verdana-Bold"];
        porktalk.position = pork.position;
        porktalk.text = @"Hi, I'm Porking";
        magicbook = [SKSpriteNode spriteNodeWithImageNamed:@"magicbookicon"];
        magicbook.position = CGPointMake(-magicbook.size.width/2, 230);
        [pork setScale:0.8];
        SKSpriteNode *wizardIcon = [SKSpriteNode
spriteNodeWithImageNamed:@"wizardicon"];
        wizardIcon.position = CGPointMake(100, 80);
        wizardIcon.name = @"wizardIcon";
        [wizardIcon setScale:0.15];
        SKSpriteNode *witchIcon = [SKSpriteNode
spriteNodeWithImageNamed:@"witchicon"];
        witchIcon.position = CGPointMake(500, 200);
        witchIcon.name = @"witchIcon";
        [witchIcon setScale:0.8];
        SKLabelNode *dialogue1 = [SKLabelNode labelNodeWithFontNamed:@"Verdana-
Bold"];
        dialogue1.text = [NSString stringWithFormat:@"%@", content1];
        dialogue1.name = @"dialogue1";
        dialogue1.fontSize = 14;
        dialogue1.horizontalAlignmentMode = SKLabelHorizontalAlignmentModeRight;
        dialogue1.position = CGPointMake(420, 250);

        SKLabelNode *dialogue2 = [SKLabelNode labelNodeWithFontNamed:@"Verdana-
Bold"];
        dialogue2.text = [NSString stringWithFormat:@"%@", content1];
        dialogue2.name = @"dialogue2";
        dialogue2.fontSize = 14;
        dialogue2.horizontalAlignmentMode = SKLabelHorizontalAlignmentModeLeft;
        dialogue2.position = CGPointMake(200, 50);

        SKSpriteNode *dialogueright = [SKSpriteNode
spriteNodeWithImageNamed:@"dialogueright" ];
        dialogueright.position = CGPointMake(230, 250);
        //[dialogueright setScale:0.12];
        [dialogueright setSize:CGSizeMake(480, 150)];
        SKSpriteNode *dialogueleft = [SKSpriteNode
spriteNodeWithImageNamed:@"dialogueleft" ];
        dialogueleft.position = CGPointMake(300, 70);
        [dialogueleft setScale:0.12];
        i = 0;
```

```
        [self addChild:bg];
        [self addChild:wizardIcon];
        [self addChild:witchIcon];
        [self addChild:dialogueright];
        [self addChild:dialogueleft];
        [self addChild:magicbook];
        [self addChild:dialogue1];
        [self addChild:dialogue2];
        [self addChild:pork];
        [self addChild:porktalk];
    }

    return self;
}


-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    /* Called when a touch begins */

    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInNode:self];
    SKNode *node = [self nodeAtPoint:location];
    //NSLog(@"%@", node.name);
    SKTransition *reveal = [SKTransition fadeWithDuration:2];
    if ([node.name isEqualToString:@"BackButton"]) {
        MTStartUpScene *scene1= [MTStartUpScene
sceneWithSize:self.view.bounds.size];
        scene1.scaleMode = SKSceneScaleModeResizeFill;
        [self.view presentScene:scene1 transition:reveal];
    }
    if (i <= counter-1)
    {
        //NSLog(@"%d", i);
        if ([conversation[i] isEqualToString:@"switchtowizard"]) {
            i = i + 1;
            changeto = 2;
        }
        if ([conversation[i] isEqualToString:@"switchtowitch"]) {
            i = i + 1;
            changeto = 1;
        }
        if ([conversation[i] isEqualToString:@"over"]) {
            MTStageSelectScene *scene1= [MTStageSelectScene
sceneWithSize:self.view.bounds.size];
            scene1.scaleMode = SKSceneScaleModeResizeFill;
            [self.view presentScene:scene1 transition:reveal];
            return;
        }
        if (changeto == 1){
            SKLabelNode *dialogue1 = (SKLabelNode*)[self
childNodeWithName:@"dialogue1"];
            content1 = conversation[i];
```

```
            dialogue1.text = [NSString stringWithFormat:@"%@", content1];
            if ([conversation[i] isEqualToString:@"He looks like this..."]) {
                [pork runAction:[SKAction moveTo:CGPointMake(500, 80) duration:0.2]];
                [porktalk runAction:[SKAction moveTo:CGPointMake(400, 80) duration:0.2]];
            }
            if ([conversation[i] isEqualToString:@"I will award you the book of Magic..."]) {
                [magicbook runAction:[SKAction moveTo:CGPointMake(100, 230)
duration:0.4]];
            }

            i++;
        }
        if (changeto == 2){
            SKLabelNode *dialogue2 = (SKLabelNode*)[self
childNodeWithName:@"dialogue2"];
            content1 = conversation[i];
            dialogue2.text = [NSString stringWithFormat:@"%@", content1];
            i++;
        }
    }

}

@end



//
//  MTStageSelectScene.h
//  magictale
//
//  Created by Yetian Mao on 5/6/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>

@interface MTStageSelectScene : SKScene

@property (nonatomic) SKSpriteNode *map;
@property (nonatomic) SKSpriteNode *map1;
- (id)initWithSize:(CGSize)size;

@end



//
//  MTStageSelectScene.m
//  magictale
//
```

```
//  Created by Yetian Mao on 5/6/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTStageSelectScene.h"
#import "MTTemplateGameScene.h"
#import "MTStartUpScene.h"

@implementation MTStageSelectScene


- (id)initWithSize:(CGSize)size {
    if (self = [super initWithSize:size]) {
        SKSpriteNode *bg = [SKSpriteNode
spriteNodeWithImageNamed:@"greybackground"];
        bg.position = CGPointMake(self.size.width / 2, self.size.height / 2);
        [bg setScale:0.227];
        SKSpriteNode *leftbutton = [SKSpriteNode
spriteNodeWithImageNamed:@"LeftButton"];
        leftbutton.position = CGPointMake(50, self.size.height/2-30);
        leftbutton.name = @"LeftButton";
        [leftbutton setScale:0.3];
        SKSpriteNode *continuebutton = [SKSpriteNode
spriteNodeWithImageNamed:@"RightButton"];
        continuebutton.position = CGPointMake(520, self.size.height/2-30);
        continuebutton.name = @"RightButton";
        [continuebutton setScale:0.3];
        SKSpriteNode *returnbutton = [SKSpriteNode
spriteNodeWithImageNamed:@"returnbutton"];
        returnbutton.position = CGPointMake(30, 280);
        returnbutton.name = @"BackButton";
        [returnbutton setScale:0.2];
        SKSpriteNode *stagetitle = [SKSpriteNode
spriteNodeWithImageNamed:@"stage_title"];
        stagetitle.position = CGPointMake(self.size.width/2, self.size.height/2 + 110);
        [stagetitle setScale:0.3];

        _map = [SKSpriteNode spriteNodeWithImageNamed:@"DayTown"];
        _map.position = CGPointMake(self.size.width / 2, self.size.height / 2 - 30);
        _map.name = @"DayTown";
        [_map setScale:0.25];
        _map1 = [SKSpriteNode spriteNodeWithImageNamed:@"NightTown"];
        [_map1 setScale:0.25];
        _map1.position = CGPointMake(self.size.width + _map1.size.width/2,
self.size.height/2 - 30);

        [self addChild:bg];
        [self addChild:_map];
        [self addChild:_map1];
        [self addChild:leftbutton];
        [self addChild:continuebutton];
        [self addChild:returnbutton];
```

```
        [self addChild:stagetitle];
    }

    return self;
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    /* Called when a touch begins */

    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInNode:self];
    SKNode *node = [self nodeAtPoint:location];
    SKTransition *reveal = [SKTransition fadeWithDuration:1];
    //NSLog(@"%@", node.name);
    if ([node.name isEqualToString:@"DayTown"] || [node.name
isEqualToString:@"NightTown"]) {
        MTTemplateGameScene *scene = [MTTemplateGameScene
sceneWithSize:self.view.bounds.size];
        scene.scaleMode = SKSceneScaleModeResizeFill;
        [self.view presentScene:scene transition:reveal];
    }
    if ([node.name isEqualToString:@"LeftButton"]) {
        [_map1 runAction:[SKAction moveTo:CGPointMake(self.size.width/2,
self.size.height/2 - 30) duration:0.2]];
        [_map runAction:[SKAction moveTo:CGPointMake(-_map.size.width/2,
self.size.height/2 - 30) duration:0.2]];
    }
    if ([node.name isEqualToString:@"RightButton"]) {
        [_map runAction:[SKAction moveTo:CGPointMake(self.size.width/2,
self.size.height/2 - 30) duration:0.2]];
        [_map1 runAction:[SKAction moveTo:CGPointMake(self.size.width +
_map.size.width/2, self.size.height/2 - 30)
                                 duration:0.2]];
    }
    if ([node.name isEqualToString:@"BackButton"]) {
        MTStartUpScene *scene1= [MTStartUpScene
sceneWithSize:self.view.bounds.size];
        scene1.scaleMode = SKSceneScaleModeResizeFill;
        [self.view presentScene:scene1 transition:reveal];
    }

}


@end


//
//  MTStartUpScene.h
//  MagicTale
//
//  Created by Lu Cao on 1/27/14.
```

```
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>
#import "MTTemplateGameScene.h"

@interface MTStartUpScene : SKScene

- (id)initWithSize:(CGSize)size;

@end


//
//  MTStartUpScene.m
//  MagicTale
//
//  Created by Lu Cao on 1/27/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTStartUpScene.h"
#import "MTTemplateGameScene.h"
#import "MTStageSelectScene.h"
#import "MTSetting.h"
#import "MTCutScene.h"

@implementation MTStartUpScene

- (id)initWithSize:(CGSize)size {
    if (self = [super initWithSize:size]) {
//        [SKTexture preloadTextures:testArray withCompletionHandler:^
//         {
//             NSLog(@"finish loading");
//         }];
        SKSpriteNode *bg = [SKSpriteNode spriteNodeWithImageNamed:@"Background"];
        bg.position = CGPointMake(self.size.width / 2, self.size.height / 2);
        NSLog(@"height:%f", self.size.height/2);
        NSLog(@"width:%f", self.size.width/2);
        [bg setScale:0.227];
        SKSpriteNode *playbutton = [SKSpriteNode spriteNodeWithImageNamed:@"Play"];
        playbutton.position = CGPointMake(60, 30);
        playbutton.name = @"playbutton";
        [playbutton setScale:0.2];
        SKSpriteNode *continuebutton = [SKSpriteNode
spriteNodeWithImageNamed:@"Continue.png"];
        continuebutton.position = CGPointMake(190, 30);
        continuebutton.name = @"continuebutton";
        [continuebutton setScale:0.2];
        SKSpriteNode *settingbutton = [SKSpriteNode
spriteNodeWithImageNamed:@"Setting"];
        settingbutton.position = CGPointMake(323, 30);
```

```
        settingbutton.name = @"settingbutton";
        [settingbutton setScale:0.2];
        SKSpriteNode *gametitle = [SKSpriteNode
spriteNodeWithImageNamed:@"gametitle"];
        gametitle.position = CGPointMake(200, 200);
        //gametitle.colorBlendFactor = 0.2;
        [gametitle setScale:0.4];

        [self addChild:bg];
        [self addChild:playbutton];
        [self addChild:continuebutton];
        [self addChild:settingbutton];
        [self addChild:gametitle];
    }

    return self;
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    /* Called when a touch begins */

    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInNode:self];
    SKNode *node = [self nodeAtPoint:location];
    //NSLog(@"%@", node.name);
    SKTransition *reveal = [SKTransition fadeWithDuration:2];
    if ([node.name isEqualToString:@"playbutton"]) {

        MTCutScene *scene = [MTCutScene sceneWithSize:self.view.bounds.size];
        scene.scaleMode = SKSceneScaleModeResizeFill;
        [self.view presentScene:scene transition:reveal];
    }
    if ([node.name isEqualToString:@"continuebutton"]) {

        MTStageSelectScene *scene = [MTStageSelectScene
sceneWithSize:self.view.bounds.size];
        scene.scaleMode = SKSceneScaleModeResizeFill;
        [self.view presentScene:scene transition:reveal];
    }
    if ([node.name isEqualToString:@"settingbutton"]) {

        MTSetting *scene = [MTSetting sceneWithSize:self.view.bounds.size];
        scene.scaleMode = SKSceneScaleModeResizeFill;
        [self.view presentScene:scene transition:reveal];
    }

}

@end


//
```

```
//  MTTemplateLevel.h
//  MagicTale
//
//  Created by Lu Cao on 1/14/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>
#import "JSTileMap.h"
#import "MTHero.h"
#import "MTMinion.h"

@interface MTTemplateLevel : SKNode

@property (nonatomic) JSTileMap *map;
@property (readonly,nonatomic) SKSpriteNode *bgLayer;
@property (readonly,nonatomic) SKSpriteNode *charLayer;

@property (nonatomic) MTHero *player;
@property (nonatomic) MTMinion *minion1;
@property (nonatomic) MTMinion *porc;

- (id)initWithLevel:(NSString *)level;

@end


//
//  MTTemplateLevel.m
//  MagicTale
//
//  Created by Lu Cao on 1/14/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTTemplateLevel.h"

@implementation MTTemplateLevel {

}

- (id)initWithLevel:(NSString *)currentLevel {
    if (self = [super init]) {

        _map = [JSTileMap mapNamed:[currentLevel stringByAppendingString:@".tmx"]];
        [self createWorld];
    }
    return self;
}

- (void)createWorld {
    self.name = @"World";
```

```
    // create background layer
    _bgLayer = [SKSpriteNode node];
    [self addChild:_bgLayer];

    // setup map
    [self setupMap];

    // setup collision areas
    [self createCollisionAreas];

    // create character layer
    _charLayer = [SKSpriteNode node];
    _charLayer.name = @"Characters";
    [self addChild:_charLayer];

    // setup player
    [self setupPlayer];

    // setup enemy
    [self setupEnemy];
}

- (void)setupMap {

    TMXLayer *layer = [_map layerNamed:@"Background"];
    CGSize size = CGSizeMake(layer.layerWidth, layer.layerHeight);
    _bgLayer.size = size;
    [_bgLayer addChild:_map];
}

- (void)createCollisionAreas {

    TMXObjectGroup *groupRock = [_map groupNamed:@"Rock"];

    NSArray *rockObjects = [groupRock objectsNamed:@"rock"];
    for (NSDictionary *rockObj in rockObjects) {
        CGFloat x = [rockObj[@"x"] floatValue];
        CGFloat y = [rockObj[@"y"] floatValue];
        CGFloat w = [rockObj[@"width"] floatValue];
        CGFloat h = [rockObj[@"height"] floatValue];

       // NSLog(@"x = %f", x);
        SKSpriteNode* rock = [SKSpriteNode spriteNodeWithColor:[SKColor redColor]
size:CGSizeMake(w, h)];

        rock.name = @"rock";
        rock.position = CGPointMake(x + w/2, y + h/2);

        rock.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:CGSizeMake(w, h)];
        rock.physicsBody.dynamic = NO;
        rock.physicsBody.categoryBitMask = MTColliderTypeRock;
```

```
        rock.physicsBody.collisionBitMask = MTColliderTypeEnemy | MTColliderTypeHero;
        rock.physicsBody.contactTestBitMask = MTColliderTypeSpell;
        rock.hidden = YES;

        [self addChild:rock];
    }

    TMXObjectGroup *groupTree = [_map groupNamed:@"Tree"];

    NSArray *treeObjects = [groupTree objectsNamed:@"tree"];
    for (NSDictionary *treeObj in treeObjects) {
        CGFloat x = [treeObj[@"x"] floatValue];
        CGFloat y = [treeObj[@"y"] floatValue];
        CGFloat w = [treeObj[@"width"] floatValue];
        CGFloat h = [treeObj[@"height"] floatValue];

        // NSLog(@"x = %f", x);
        SKSpriteNode* tree = [SKSpriteNode spriteNodeWithColor:[SKColor redColor]
size:CGSizeMake(w, h)];

        tree.name = @"tree";
        tree.position = CGPointMake(x + w/2, y + h/2);

        tree.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:CGSizeMake(w, h)];
        tree.physicsBody.dynamic = NO;
        tree.physicsBody.categoryBitMask = MTColliderTypeTree;
        tree.physicsBody.collisionBitMask = MTColliderTypeEnemy | MTColliderTypeHero;
        tree.physicsBody.contactTestBitMask = MTColliderTypeSpell;
        tree.hidden = YES;

        [self addChild:tree];
    }

    TMXObjectGroup *groupWall = [_map groupNamed:@"Wall"];

    NSArray *wallObjects = [groupWall objectsNamed:@"wall"];
    for (NSDictionary *wallObj in wallObjects) {
        CGFloat x = [wallObj[@"x"] floatValue];
        CGFloat y = [wallObj[@"y"] floatValue];
        CGFloat w = [wallObj[@"width"] floatValue];
        CGFloat h = [wallObj[@"height"] floatValue];

        // NSLog(@"x = %f", x);
        SKSpriteNode* wall = [SKSpriteNode spriteNodeWithColor:[SKColor redColor]
size:CGSizeMake(w, h)];

        wall.name = @"wall";
        wall.position = CGPointMake(x + w/2, y + h/2);

        wall.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:CGSizeMake(w, h)];
        wall.physicsBody.dynamic = NO;
        wall.physicsBody.categoryBitMask = MTColliderTypeWall;
```

```
        wall.physicsBody.collisionBitMask = MTColliderTypeEnemy | MTColliderTypeHero;
        wall.physicsBody.contactTestBitMask = MTColliderTypeSpell;
        wall.hidden = YES;

        [self addChild:wall];
    }

    TMXObjectGroup *groupWater = [_map groupNamed:@"Water"];

    NSArray *waterObjects = [groupWater objectsNamed:@"water"];
    for (NSDictionary *waterObj in waterObjects) {
        CGFloat x = [waterObj[@"x"] floatValue];
        CGFloat y = [waterObj[@"y"] floatValue];
        CGFloat w = [waterObj[@"width"] floatValue];
        CGFloat h = [waterObj[@"height"] floatValue];

        // NSLog(@"x = %f", x);
        SKSpriteNode* water = [SKSpriteNode spriteNodeWithColor:[SKColor redColor]
size:CGSizeMake(w, h)];

        water.name = @"water";
        water.position = CGPointMake(x + w/2, y + h/2);

        water.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:CGSizeMake(w,
h)];
        water.physicsBody.dynamic = NO;
        water.physicsBody.categoryBitMask = MTColliderTypeWater;
        water.physicsBody.collisionBitMask = MTColliderTypeEnemy | MTColliderTypeHero;
        water.physicsBody.contactTestBitMask = MTColliderTypeSpell;
        water.hidden = YES;

        [self addChild:water];
    }

    TMXObjectGroup *groupHouse = [_map groupNamed:@"House"];

    NSArray *houseObjects = [groupHouse objectsNamed:@"house"];
    for (NSDictionary *houseObj in houseObjects) {
        CGFloat x = [houseObj[@"x"] floatValue];
        CGFloat y = [houseObj[@"y"] floatValue];
        CGFloat w = [houseObj[@"width"] floatValue];
        CGFloat h = [houseObj[@"height"] floatValue];

        // NSLog(@"x = %f", x);
        SKSpriteNode* house = [SKSpriteNode spriteNodeWithColor:[SKColor redColor]
size:CGSizeMake(w, h)];

        house.name = @"house";
        house.position = CGPointMake(x + w/2, y + h/2);

        house.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:CGSizeMake(w,
h)];
```

```
        house.physicsBody.dynamic = NO;
        house.physicsBody.categoryBitMask = MTColliderTypeHouse;
        house.physicsBody.collisionBitMask = MTColliderTypeEnemy |
MTColliderTypeHero;
        house.physicsBody.contactTestBitMask = MTColliderTypeSpell;
        house.hidden = YES;

        [self addChild:house];
    }

    TMXObjectGroup *groupFence = [_map groupNamed:@"Fence"];

    NSArray *fenceObjects = [groupFence objectsNamed:@"fence"];
    for (NSDictionary *fenceObj in fenceObjects) {
        CGFloat x = [fenceObj[@"x"] floatValue];
        CGFloat y = [fenceObj[@"y"] floatValue];
        CGFloat w = [fenceObj[@"width"] floatValue];
        CGFloat h = [fenceObj[@"height"] floatValue];

        // NSLog(@"x = %f", x);
        SKSpriteNode* fence = [SKSpriteNode spriteNodeWithColor:[SKColor redColor]
size:CGSizeMake(w, h)];

        fence.name = @"fence";
        fence.position = CGPointMake(x + w/2, y + h/2);

        fence.physicsBody = [SKPhysicsBody bodyWithRectangleOfSize:CGSizeMake(w,
h)];
        fence.physicsBody.dynamic = NO;
        fence.physicsBody.categoryBitMask = MTColliderTypeFence;
        fence.physicsBody.collisionBitMask = MTColliderTypeEnemy | MTColliderTypeHero;
        fence.physicsBody.contactTestBitMask = MTColliderTypeSpell;
        fence.hidden = YES;

        [self addChild:fence];
    }
}

- (void)setupPlayer {

    TMXObjectGroup *group = [_map groupNamed:@"Characters"];
    NSDictionary *playerObj = [group objectNamed:@"player"];
    CGPoint position = CGPointMake([playerObj[@"x"] floatValue], [playerObj[@"y"]
floatValue]);
    _player = [[MTHero alloc] initAtPosition:position];
    _player.spawnPosition = position;
    NSLog(@"%f", position.x);
    [_player setScale:0.42];
    [_charLayer addChild:_player];
}

- (void)setupEnemy {
```

```
    TMXObjectGroup *group = [_map groupNamed:@"Characters"];
    NSArray *enemyObjects = [group objectsNamed:@"Slime"];
    for (NSDictionary *enemyObj in enemyObjects) {
        CGFloat x = [enemyObj[@"x"] floatValue];
        CGFloat y = [enemyObj[@"y"] floatValue];
        _minion1 = [[MTMinion alloc] initWithName:@"Slime" atPosition:CGPointMake(x,
y)];
        [_minion1 setScale:0.3];
        [_charLayer addChild:_minion1];
    }

    self.porc = [[MTMinion alloc] initWithName:@"Porcr" atPosition:CGPointMake(300,
500)];
    [self.porc setScale:0.3];
    [_charLayer addChild:self.porc];

}

@end



//
//  MTTemplateGameScene.h
//  MagicTale
//
//  Created by Lu Cao on 1/14/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>
#import "MTTemplateLevel.h"
#import "MTUI.h"

@interface MTTemplateGameScene : SKScene

@property (nonatomic) NSString *currentLevel;

@property (nonatomic) NSTimeInterval lastUpdateTimeInterval; // the previous update:
loop time interval

@property (nonatomic) MTTemplateLevel *world;
@property (nonatomic) MTUI *ui;

- (id)initWithSize:(CGSize)size;

@end


//
//  MTTemplateGameScene.m
```

```
//  MagicTale
//
//  Created by Lu Cao on 1/14/14.
//  Copyright (c) 2014 Lu. All rights reserved.
//

#import "MTTemplateGameScene.h"

@interface MTTemplateGameScene () <SKPhysicsContactDelegate>
@end

@implementation MTTemplateGameScene {
    BOOL _touchSequence;   // check if begin:1, move:2, and end:3 is one touch
}

- (id)initWithSize:(CGSize)size{
    if (self = [super initWithSize:size]) {
        [self buildWorldForCurrentLevel];
        [self buildUI];

        //
        _world.player.selectedSpellType = MTSpellTypeFireball;
    }

    return self;
}

- (void)buildWorldForCurrentLevel {
    self.anchorPoint = CGPointMake(0.5, 0.5);
    self.physicsWorld.gravity = CGVectorMake(0, 0);
    self.physicsWorld.contactDelegate = self;


    // set up worldNode, the root Node
    _currentLevel = @"demo-level";
    _world = [[MTTemplateLevel alloc] initWithLevel:_currentLevel];
//    _world.position = CGPointMake(CGRectGetMidX(self.frame),
CGRectGetMidY(self.frame));
    [self addChild:_world];
}

- (void)buildUI {
    _ui = [[MTUI alloc] init];
    _ui.position = CGPointMake(CGRectGetMidX(self.frame),
CGRectGetMidY(self.frame));
    [self addChild:_ui];
}

- (void)centerViewOn:(CGPoint)centerOn {
    CGSize size = self.size;
    CGFloat x = Clamp(centerOn.x, size.width / 2, _world.bgLayer.size.width - size.width /
2);
```

```
    CGFloat y = Clamp(centerOn.y, size.height / 2, _world.bgLayer.size.height -
size.height/2);
    _world.position = CGPointMake(-x, -y);
}


- (void)update:(NSTimeInterval)currentTime {
    // Handle time delta.
    // If we drop below 60fps, we still want everything to move the same distance.
    CFTimeInterval timeSinceLast = currentTime - self.lastUpdateTimeInterval;
    self.lastUpdateTimeInterval = currentTime;
    if (timeSinceLast > 1) { // more than a second since last update
        timeSinceLast = 1.0f / 60.0f;
        self.lastUpdateTimeInterval = currentTime;
    }


    [_world.player updateWithTimeSinceLastUpdate:timeSinceLast];
    [_world.minion1 updateWithTimeSinceLastUpdate:timeSinceLast];
    [_world.porc updateWithTimeSinceLastUpdate:timeSinceLast];

    // move player
    if (_world.player.moveRequested && _world.player.canMove) {
        if (MTDistanceBetweenPoints(_world.player.userTouchedPosition,
_world.player.position) < 1) {
            _world.player.requestedAnimation = MTAnimationStateIdle;
            _world.player.moveRequested = NO;
        } else if (_world.player.moveRequested) {
            [_world.player moveTowards:_world.player.userTouchedPosition
withTimeInterval:timeSinceLast];
            _world.player.requestedAnimation = MTAnimationStateWalk;
        }
    }


    [_ui setBarProgress:_world.player];

    /*
    if (_world.player.isHoldingSpell) {
          NSLog(@"is");
    } else NSLog(@"NO");
     */
}

- (void)didSimulatePhysics {
    [self centerViewOn:_world.player.position];
}

#pragma mark - collision detection
- (void)didBeginContact:(SKPhysicsContact *)contact {
    // Either bodyA or bodyB in the collision could be a character.
    SKNode *nodeA = contact.bodyA.node;
```

```
    SKNode *nodeB = contact.bodyB.node;

    // when a projectile kinda spell hit wall, remove it
    if ([nodeA isKindOfClass:[MTSpellNode class]])
        if (nodeB.physicsBody.categoryBitMask == MTColliderTypeWall)
            if (((MTSpellNode *)nodeA).castType == MTCastTypeProjectile)
                [nodeA removeFromParent];
    if ([nodeB isKindOfClass:[MTSpellNode class]])
        if (nodeA.physicsBody.categoryBitMask == MTColliderTypeWall)
            if (((MTSpellNode *)nodeB).castType == MTCastTypeProjectile)
                [nodeB removeFromParent];

    if ([nodeA.name isEqualToString:@"enemyProjectile"])
        NSLog(@"HIT");
    if ([nodeB.name isEqualToString:@"enemyProjectile"])
        NSLog(@"HIT");

    /*
    // Check bodyA
    if ([nodeA isKindOfClass:[MTSpellNode class]]) {
        MTSpellNode *nodea = (MTSpellNode *)nodeA;
        if ([nodeB isKindOfClass:[MTEnemy class]]) {
            MTEnemy *nodeb = (MTEnemy *)nodeB;
            [nodeb applyDamageFromSpell:nodea];
            [nodea removeFromParent];
        }
    }

    // Check bodyB
    if ([nodeB isKindOfClass:[MTSpellNode class]]) {
        MTSpellNode *nodeb = (MTSpellNode *)nodeB;
        if ([nodeA isKindOfClass:[MTEnemy class]]) {
            MTEnemy *nodea = (MTEnemy *)nodeA;
            [nodea applyDamageFromSpell:nodeb];
            [nodeb removeFromParent];
        }
    }
     */
}

#pragma mark - Touch reactions

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    _touchSequence = NO;

    CGPoint _touchedLocation = [touch locationInNode:_ui];

    SKSpriteNode *touchedNode = (SKSpriteNode *)[_ui nodeAtPoint:_touchedLocation];
    //NSLog(@"%@", touchedNode.name);
    if ([touchedNode.name isEqualToString:@"magicBook"]) {
```

```
        //NSLog(@"111");
        [_ui showMagicCircle];
    }

    _world.player.userTouchedPosition = [touch locationInNode:_world];
    if (_world.player.canInteract) {
        _world.player.moveRequested = YES;
//        _world.player.castRequested = YES;
        _world.player.userTouchedPosition = [touch locationInNode:_world];
        _world.player.requestedAnimation = MTAnimationStateWalk;
    }


}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    if (_world.player.moveRequested) {
//        _world.player.moveRequested = YES;
        _world.player.userTouchedPosition = [touch locationInNode:_world];
//        _world.player.requestedAnimation = MTAnimationStateWalk;
    }

    if (_world.player.castingStates && _touchSequence) {
        [_world.player faceTo:[touch locationInNode:_world]];
        _world.player.castingStates = 2;
    }
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    _touchSequence = NO;

    if (_world.player.castingStates == 2) {
        _world.player.userTouchedPosition = [touch locationInNode:_world];
        _world.player.requestedAnimation = MTAnimationStatePostAttack;
        _world.player.castingStates = 0;
    }
}


@end


//
//  MTSpellNode.h
//  MagicTale
//
//  Created by Lu Cao on 12/28/13.
//  Copyright (c) 2013 Lu. All rights reserved.
```

```
//

/* Define spell type */
typedef enum {
    MTSpellTypeNoMana = 0,
    MTSpellTypeFireball,
    MTSpellTypeIceSpear,
    MTSpellTypeBlizzard,
    MTSpellTypeTeleport,
} MTSpellType;

typedef enum : uint8_t {
    MTCastTypeProjectile = 0,
    MTCastTypeLocation,
    MTCastTypeSelf,
} MTCastType;

#import "MTCharacter.h"
#import "MTUKit.h"

@interface MTSpellNode : SKSpriteNode

// basic attributes
@property (nonatomic) CGFloat spellFlyingSpeed;
@property (nonatomic) CGFloat spellLifeTime;
@property (nonatomic) CGFloat damage;
@property (nonatomic) CGFloat cost;
@property (nonatomic) MTSpellType spellType;
@property (nonatomic) MTCastType castType;
@property (nonatomic) SKEmitterNode *spellEmitter;

// status the spell will apply
@property (nonatomic) MTAbnormalStatus status;
@property (nonatomic) CGFloat statusDuration;
@property (nonatomic) CGFloat statusDamage;

// initialization
- (id)initAtPosition:(CGPoint)position
       withSpellType:(MTSpellType)spellType
   andDamageModifier:(CGFloat)damageModifier;

// ready up and casting spell
- (void)readySpellWithPosition:(CGPoint)position;
- (void)castSpellWithPosition:(CGPoint)position;

@end




//
```

```
//  MTSpellNode.m
//  MagicTale
//
//  Created by Lu Cao on 12/28/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTSpellNode.h"
#import "MTCharacter.h"

@implementation MTSpellNode

#pragma mark - Initialization
- (id)initAtPosition:(CGPoint)position
       withSpellType:(MTSpellType)spellType
   andDamageModifier:(CGFloat)damageModifier {
    self = [super init];
    if (self) {
        // init attributes
        self.name = @"SpellNode";
        self.position = position;
        self.size = CGSizeMake(1, 1);
        self.anchorPoint = CGPointMake(0.5, 0.5);

        [self resolveSpellType:spellType];

        self.physicsBody = [SKPhysicsBody bodyWithCircleOfRadius:self.size.width/2];
        [self configurePhysics];

        [self debugDraw];
    }

    return self;
}

- (void)configurePhysics {
    self.physicsBody.categoryBitMask = MTColliderTypeSpell;
    if (self.castType == MTCastTypeProjectile)
        self.physicsBody.collisionBitMask = MTColliderTypeWall | MTColliderTypeEnemy;
    else
        self.physicsBody.collisionBitMask = 0;
    self.physicsBody.contactTestBitMask = MTColliderTypeEnemy | MTColliderTypeWall;
}

- (void)resolveSpellType:(MTSpellType)spellType {
    switch (spellType) {

        case MTSpellTypeNoMana: {
            self.spellFlyingSpeed = 65;
            self.spellLifeTime = 1;
            self.damage = 0;
            self.cost = 0;
```

```
            self.size = CGSizeMake(1, 1);
            self.spellEmitter = [NSKeyedUnarchiver unarchiveObjectWithFile:[[NSBundle
mainBundle] pathForResource:@"NoMana" ofType:@"sks"]];
            self.castType = MTCastTypeProjectile;
        }
            break;

        case MTSpellTypeFireball: {
            self.spellFlyingSpeed = 300;
            self.spellLifeTime = 2;
            self.damage = 34;
            self.cost = 10;
            self.status = MTAbnormalStatusBurning;
            self.statusDuration = 2;
            self.statusDamage = self.damage * 0.1;
            self.size = CGSizeMake(10, 10);
            self.spellEmitter = [NSKeyedUnarchiver unarchiveObjectWithFile:[[NSBundle
mainBundle] pathForResource:@"Fireball" ofType:@"sks"]];
            self.castType = MTCastTypeProjectile;
        }
            break;

        case MTSpellTypeIceSpear: {
            self.spellFlyingSpeed = 180;
            self.spellLifeTime = 5;
            self.damage = 8;
            self.cost = 15;
            self.status = MTAbnormalStatusChilled;
            self.statusDuration = 5;
            self.statusDamage = 0;
            self.size = CGSizeMake(12, 12);
            self.spellEmitter = [NSKeyedUnarchiver unarchiveObjectWithFile:[[NSBundle
mainBundle] pathForResource:@"IceSpear" ofType:@"sks"]];
            self.castType = MTCastTypeProjectile;
        }
            break;

        case MTSpellTypeBlizzard: {
            self.spellFlyingSpeed = 100;
            self.spellLifeTime = 5;
            self.damage = 30;
            self.cost = 25;
            self.status = MTAbnormalStatusChilled;
            self.statusDuration = 1;
            self.statusDamage = 0;
            self.size = CGSizeMake(80, 80);
            self.spellEmitter = [NSKeyedUnarchiver unarchiveObjectWithFile:[[NSBundle
mainBundle] pathForResource:@"Blizzard" ofType:@"sks"]];
            self.castType = MTCastTypeLocation;
        }
            break;
```

```
        case MTSpellTypeTeleport: {
            self.spellFlyingSpeed = 1;
            self.spellLifeTime = 1;
            self.damage = 0;
            self.cost = 30;
            self.size = CGSizeMake(1, 1);
            self.spellEmitter = [NSKeyedUnarchiver unarchiveObjectWithFile:[[NSBundle
mainBundle] pathForResource:@"NoMana" ofType:@"sks"]];
            self.castType = MTCastTypeLocation;
        }
            break;

        default:
            break;
    }
}

// ready spell at hero's position
- (void)readySpellWithPosition:(CGPoint)position {

    self.spellEmitter.name = @"SpellEmitter";
    [self addChild:self.spellEmitter];

    self.spellEmitter.position = CGPointZero;
    self.spellEmitter.targetNode = [self parent];
    self.hidden = YES;
}

// cast spell out
- (void)castSpellWithPosition:(CGPoint)position {

    self.hidden = NO;
    CGFloat rot = [[self parent] childNodeWithName:@"Player"].zRotation;

    switch (self.castType) {
        case MTCastTypeProjectile:
            [self runAction:[SKAction moveByX:-
sinf(rot)*self.spellFlyingSpeed*self.spellLifeTime
                                        y:cosf(rot)*self.spellFlyingSpeed*self.spellLifeTime
                                 duration:self.spellLifeTime]];
            [self runAction:[SKAction sequence:@[[SKAction
waitForDuration:self.spellLifeTime],
                                                 //[SKAction fadeOutWithDuration:1],
                                                 [SKAction removeFromParent]]]];
            break;

        case MTCastTypeLocation:
            if (self.spellType == MTSpellTypeTeleport) {
                self.position = [[self parent] childNodeWithName:@"Player"].position;
                [[self parent] childNodeWithName:@"Player"].position = position;
                [self runAction:[SKAction sequence:@[[SKAction
waitForDuration:self.spellLifeTime],
```

```
                                        [SKAction removeFromParent]]];
        } else {
            self.position = position;
            [self runAction:[SKAction sequence:@[[SKAction
waitForDuration:self.spellLifeTime],
                                        [SKAction removeFromParent]]];
        }
        break;

    default:
        break;
  }
}

// draw a outline for physical body, debug
- (void)debugDraw {
  CGRect box = CGRectMake(0, 0, self.size.width, self.size.height);
  UIBezierPath *circlePath = [UIBezierPath bezierPathWithOvalInRect:box];

  SKShapeNode *circle = [SKShapeNode node];
  circle.path = circlePath.CGPath;
  circle.strokeColor = [SKColor redColor];
  circle.position = CGPointMake(-self.size.width/2, -self.size.height/2);
  [self addChild:circle];
}

@end


//
//  MTFloatingDamage.h
//  MagicTale
//
//  Created by Lu Cao on 12/27/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>

@interface MTFloatingDamage : SKLabelNode

/* Generate a text floating damage counter going up and fade out */
- (id)initAtPosition:(CGPoint)position ofSprite:(SKSpriteNode *)sprite
withDamage:(CGFloat)damage;

/* Animation that move up and fade out */
- (void)anim;

@end


//
```

```
//  MTFloatingDamage.m
//  MagicTale
//
//  Created by Lu Cao on 12/27/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTFloatingDamage.h"

@implementation MTFloatingDamage

#pragma mark - Initialization
- (id)initAtPosition:(CGPoint)position ofSprite:(SKSpriteNode *)sprite
withDamage:(CGFloat)damage {
    if (self = [super init]) {
        self.position = sprite.position;
        self.text = [NSString stringWithFormat:@"%i", (int) damage];
    }

    return self;
}

#pragma mark - Animation
- (void)anim {
    [self runAction:[SKAction sequence:@[[SKAction group:@[[SKAction moveByX:0 y:30
duration:2],
                                                           [SKAction fadeOutWithDuration:2]]],
                                    [SKAction runBlock:^{[self removeFromParent];}]]]];
}

@end


//
//  MTUI.h
//  MagicTale
//
//  Created by Lu Cao on 12/25/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import <SpriteKit/SpriteKit.h>
#import "MTHero.h"

@interface MTUI : SKSpriteNode

/* Define UI sprites */
//@property (nonatomic) SKSpriteNode *hpBar;
//@property (nonatomic) SKSpriteNode *mpBar;
//@property (nonatomic) SKSpriteNode *expBar;
@property (nonatomic) SKSpriteNode *hp;
@property (nonatomic) SKSpriteNode *mp;
```

```
@property (nonatomic) SKSpriteNode *exp;
@property (nonatomic) SKSpriteNode *hpContainer;
@property (nonatomic) SKSpriteNode *mpContainer;
@property (nonatomic) SKSpriteNode *expContainer;
@property (nonatomic) SKSpriteNode *magicBook;              // magic book is used for
casting magic by click
@property (nonatomic) SKSpriteNode *magicCircle;
@property (nonatomic) NSMutableArray *magicCircleNodes;
@property (nonatomic) SKSpriteNode *fingerTrackNode;

@property (nonatomic) SKSpriteNode *magicCircleNode1;
@property (nonatomic) SKSpriteNode *magicCircleNode2;
@property (nonatomic) SKSpriteNode *magicCircleNode3;
@property (nonatomic) SKSpriteNode *magicCircleNode4;
@property (nonatomic) SKSpriteNode *magicCircleNode5;
@property (nonatomic) SKSpriteNode *magicCircleNode6;

- (id)init;
- (void)initUI;
- (void)showMagicCircle;
- (void)hideMagicCircle;

- (void)addFingerTrackAtPosition:(CGPoint)position;
- (void)setBarProgress:(MTHero *)player;

- (void)updateUI;

@end


//
//  MTUI.m
//  MagicTale
//
//  Created by Lu Cao on 12/25/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTUI.h"

@implementation MTUI

- (id)init {
    if (self = [super init]) {

//        self.size = CGSizeMake(1130, 640);
        self.anchorPoint = CGPointMake(0, 0);
        [self initUI];

    }

    return self;
```

```
}

- (void)initUI {
    [self initMagicBook];
    [self initMagicCircle];
    [self initBars];
}

- (void)initBars {

    // set up hp container
    self.hpContainer = [SKSpriteNode spriteNodeWithImageNamed:@"container"];
    self.hpContainer.name = @"hpContainer";
    self.hpContainer.size = CGSizeMake(180, 120);
    self.hpContainer.position = CGPointMake(-215, -100);
    self.hp = [SKSpriteNode spriteNodeWithImageNamed:@"hp"];
    self.hp.size = CGSizeMake(160, 98);
    self.hp.position = CGPointMake(0, -46);
    self.hp.anchorPoint = CGPointMake(0.5, 0);
    SKSpriteNode *hpMask = [SKSpriteNode spriteNodeWithImageNamed:@"mask"];
    hpMask.size = self.hpContainer.size;
    SKCropNode *hpCropNode = [SKCropNode node];
    [hpCropNode addChild:self.hp];
    [hpCropNode setMaskNode:hpMask];
    [self addChild: hpCropNode];
    hpCropNode.position = self.hpContainer.position;
    [self addChild:self.hpContainer];


    // set up mp container
    self.mpContainer = [SKSpriteNode spriteNodeWithImageNamed:@"container"];
    self.mpContainer.name = @"mpContainer";
    self.mpContainer.size = CGSizeMake(180, 120);
    self.mpContainer.position = CGPointMake(215, -100);
    self.mp = [SKSpriteNode spriteNodeWithImageNamed:@"mp"];
    self.mp.size = CGSizeMake(160, 120);
    self.mp.position = CGPointMake(5, -58.5);
    self.mp.anchorPoint = CGPointMake(0.5, 0);
    SKSpriteNode *mpMask = [SKSpriteNode spriteNodeWithImageNamed:@"mask"];
    mpMask.size = self.mpContainer.size;
    SKCropNode *mpCropNode = [SKCropNode node];
    [mpCropNode addChild:self.mp];
    [mpCropNode setMaskNode:mpMask];
    [self addChild: mpCropNode];
    mpCropNode.position = self.mpContainer.position;
    [self addChild:self.mpContainer];

    // set up exp container
    self.expContainer = [SKSpriteNode spriteNodeWithImageNamed:@"expContainer"];
    self.expContainer.name = @"expContainer";
    self.expContainer.size = CGSizeMake(256, 256);
    self.expContainer.position = CGPointMake(0, -125);
```

```
    self.exp = [SKSpriteNode spriteNodeWithImageNamed:@"expBar"];
    self.exp.size = CGSizeMake(235, 20);
    self.exp.position = CGPointMake(-121, 5);
    self.exp.anchorPoint = CGPointMake(0, 0.5);
    SKSpriteNode *expMask = [SKSpriteNode spriteNodeWithImageNamed:@"expMask"];
    expMask.size = self.expContainer.size;
    SKCropNode *expCropNode = [SKCropNode node];
    [expCropNode addChild:self.exp];
    [expCropNode setMaskNode:expMask];
    [self addChild: expCropNode];
    expCropNode.position = self.expContainer.position;
    [self addChild:self.expContainer];


//    SKCropNode *hpCrop = [[SKCropNode alloc] init];
//    SKShapeNode *hpMask = [[SKShapeNode alloc] init];
//    CGMutablePathRef hpCircle = CGPathCreateMutable();
//    CGPathAddArc(hpCircle, NULL, 0, 0, 42.5, 0, M_PI*2, YES);
//    hpMask.path = hpCircle;
//    hpMask.lineWidth = 0.3;
//    hpMask.strokeColor = [SKColor redColor];
//
//    [hpCrop setMaskNode:hpMask];
//    SKSpriteNode *hp = [SKSpriteNode spriteNodeWithImageNamed:@"hp"];
//    [hpCrop addChild:hp];
//    [self addChild:hpCrop];



//    // set up hp bar
//    self.hpBar = [SKSpriteNode spriteNodeWithImageNamed:@"hpBar"];
//    self.hpBar.name = @"hpBar";
//    self.hpBar.position = CGPointMake(0, 15);
//    [self addChild:self.hpBar];
//
//    // set up mp bar
//    self.mpBar = [SKSpriteNode spriteNodeWithImageNamed:@"mpBar"];
//    self.mpBar.name = @"mpBar";
//    self.mpBar.position = CGPointMake(0, 10);
//    [self addChild:self.mpBar];
//
//    // set up exp bar
//    self.expBar = [SKSpriteNode spriteNodeWithImageNamed:@"expBar"];
//    self.expBar.name = @"expBar";
//    self.expBar.position = CGPointMake(0, 5);
//    [self addChild:self.expBar];

}

- (void)initMagicBook {
    self.magicBook = [SKSpriteNode spriteNodeWithImageNamed:@"magicBookIcon"];
    self.magicBook.name = @"magicBook";
```

```
    self.magicBook.size = CGSizeMake(88, 88);
    self.magicBook.position = CGPointMake(-230, 118);
    [self addChild:self.magicBook];
}

- (void)initMagicCircle {
    // init the circle
//    self.magicCircle = [SKSpriteNode spriteNodeWithImageNamed:@"magicCircle"];
//    self.magicCircle.name = @"magicCircle";
//    [self.magicCircle setScale:0.5];
//    self.magicCircle.anchorPoint = CGPointMake(0, 0);
//    self.magicCircle.position = CGPointMake(0, 0);
//    self.magicCircle.alpha = 0;
//    [self addChild:self.magicCircle];

    SKTextureAtlas *mcAnimAtlas = [SKTextureAtlas atlasNamed:@"MagicCircle"];
    NSMutableArray *mcAnimFrames = [NSMutableArray array];
    NSInteger numImages = mcAnimAtlas.textureNames.count;
    for (int i = 1; i <= numImages; i++) {
        NSString *textureName = [NSString stringWithFormat:@"magicCircle_%d",i];
        SKTexture *temp = [mcAnimAtlas textureNamed:textureName];
        [mcAnimFrames addObject:temp];
    }
    self.magicCircle = [SKSpriteNode spriteNodeWithImageNamed:@"magicCircle_1"];
    [self addChild:self.magicCircle];
    [self.magicCircle runAction:[SKAction repeatActionForever:[SKAction
animateWithTextures:mcAnimFrames timePerFrame:0.1f]]];
    self.magicCircle.hidden = YES;

    // init the finger track emitter
    self.fingerTrackNode = [SKSpriteNode node];
    [self addChild:self.fingerTrackNode];

    // init nodes
    self.magicCircleNode1 = [SKSpriteNode spriteNodeWithColor:[SKColor yellowColor]
size:CGSizeMake(30, 30)];
    self.magicCircleNode2 = [SKSpriteNode spriteNodeWithColor:[SKColor yellowColor]
size:CGSizeMake(30, 30)];
    self.magicCircleNode3 = [SKSpriteNode spriteNodeWithColor:[SKColor yellowColor]
size:CGSizeMake(30, 30)];
    self.magicCircleNode4 = [SKSpriteNode spriteNodeWithColor:[SKColor yellowColor]
size:CGSizeMake(30, 30)];
    self.magicCircleNode5 = [SKSpriteNode spriteNodeWithColor:[SKColor yellowColor]
size:CGSizeMake(30, 30)];
    self.magicCircleNode6 = [SKSpriteNode spriteNodeWithColor:[SKColor yellowColor]
size:CGSizeMake(30, 30)];
    self.magicCircleNode1.name = @"magicCircleNode1";
    self.magicCircleNode2.name = @"magicCircleNode2";
    self.magicCircleNode3.name = @"magicCircleNode3";
    self.magicCircleNode4.name = @"magicCircleNode4";
    self.magicCircleNode5.name = @"magicCircleNode5";
    self.magicCircleNode6.name = @"magicCircleNode6";
```

```
    self.magicCircleNode1.position = CGPointMake(176, 225);
    self.magicCircleNode2.position = CGPointMake(285, 290);
    self.magicCircleNode3.position = CGPointMake(397, 225);
    self.magicCircleNode4.position = CGPointMake(397, 96);
    self.magicCircleNode5.position = CGPointMake(285, 31);
    self.magicCircleNode6.position = CGPointMake(176, 96);
    self.magicCircleNode1.alpha = 0;
    self.magicCircleNode2.alpha = 0;
    self.magicCircleNode3.alpha = 0;
    self.magicCircleNode4.alpha = 0;
    self.magicCircleNode5.alpha = 0;
    self.magicCircleNode6.alpha = 0;
    [self addChild:self.magicCircleNode1];
    [self addChild:self.magicCircleNode2];
    [self addChild:self.magicCircleNode3];
    [self addChild:self.magicCircleNode4];
    [self addChild:self.magicCircleNode5];
    [self addChild:self.magicCircleNode6];
}

- (void)addFingerTrackAtPosition:(CGPoint)position {
    SKEmitterNode *fingerTrack = [NSKeyedUnarchiver unarchiveObjectWithFile:
                  [[NSBundle mainBundle] pathForResource:@"FingerTrack"
ofType:@"sks"]];
    [self.fingerTrackNode addChild:fingerTrack];
    fingerTrack.targetNode = [self.fingerTrackNode parent];
    self.fingerTrackNode.position = position;
    [self runAction:[SKAction sequence:@[[SKAction waitForDuration:0.2],
                                [SKAction runBlock:^{[fingerTrack removeFromParent];}]]]];
}

- (void)showMagicCircle {
    [self.magicCircle runAction:[SKAction fadeAlphaTo:0.6 duration:0.5]];
}
- (void)hideMagicCircle {
    [self.magicCircle runAction:[SKAction fadeAlphaTo:0 duration:0.5]];
}

- (void)setBarProgress:(MTHero *)player {
    if (player.hp <= 0)
        [self.hp setYScale:0];
    else [self.hp setYScale:player.hp/player.maxHP];

    [self.mp setYScale:player.mp/player.maxMP];
//    [self.expBar setXScale:player.exp/player.levelUpExp];
}

- (void)updateUI {


}
```

```
@end


//
//  MTUKit.h
//  MagicTale
//
//  Created by Lu Cao on 12/24/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//
#import <CoreGraphics/CoreGraphics.h>
#import <SpriteKit/SpriteKit.h>
#import <GLKit/GLKMath.h>

#define SKT_INLINE      static __inline__


/* Generate a random float between 0.0f and 1.0f. */
#define MT_RANDOM_0_1() (arc4random() / (float)(0xffffffffu))

/* The assets are all facing Y down, so offset by pi half to get into X right facing. */
#define MT_POLAR_ADJUST(x) x + (M_PI * 0.5f)

/* Distance and coordinate utility functions. */
CGFloat MTDistanceBetweenPoints(CGPoint first, CGPoint second);
CGFloat MTRadiansBetweenPoints(CGPoint first, CGPoint second);
CGPoint MTPointByAddingCGPoints(CGPoint first, CGPoint second);

/* Load the named frames in a texture atlas into an array of frames. */
NSArray *MTLoadFramesFromAtlas(NSString *atlasName, NSString *baseFileName);

SKT_INLINE CGFloat Clamp(CGFloat value, CGFloat min, CGFloat max) {
    return value < min ? min : value > max ? max : value;
}


//
//  MTUKit.m
//  MagicTale
//
//  Created by Lu Cao on 12/24/13.
//  Copyright (c) 2013 Lu. All rights reserved.
//

#import "MTCharacter.h"

#pragma mark - Point Calculations
CGFloat MTDistanceBetweenPoints(CGPoint first, CGPoint second) {
    return hypotf(second.x - first.x, second.y - first.y);
}

CGFloat MTRadiansBetweenPoints(CGPoint first, CGPoint second) {
```

```
    CGFloat deltaX = second.x - first.x;
    CGFloat deltaY = second.y - first.y;
    return atan2f(deltaY, deltaX);
}

CGPoint MTPointByAddingCGPoints(CGPoint first, CGPoint second) {
    return CGPointMake(first.x + second.x, first.y + second.y);
}

#pragma mark - Loading from a Texture Atlas
NSArray *MTLoadFramesFromAtlas(NSString *atlasName, NSString *baseFileName) {
    SKTextureAtlas *atlas = [SKTextureAtlas atlasNamed:atlasName];
    CGFloat numOfFrames = atlas.textureNames.count;
    NSMutableArray *frames = [NSMutableArray arrayWithCapacity:numOfFrames];

    for (int i = 1; i <= numOfFrames; i++) {
        NSString *fileName = [NSString stringWithFormat:@"%@%04d.png",
baseFileName, i];
        SKTexture *texture = [atlas textureNamed:fileName];
        [frames addObject:texture];
    }

    return frames;
}
```

Santa Clara University
DEPARTMENT of COMPUTER ENGINEERING

Date: June 3, 2014

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Albert Chang, Lu Cao, Yetian Mao

ENTITLED

Mobile Game Application: The MagicTale

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THESIS ADVISOR

DEPARTMENTCHAIR