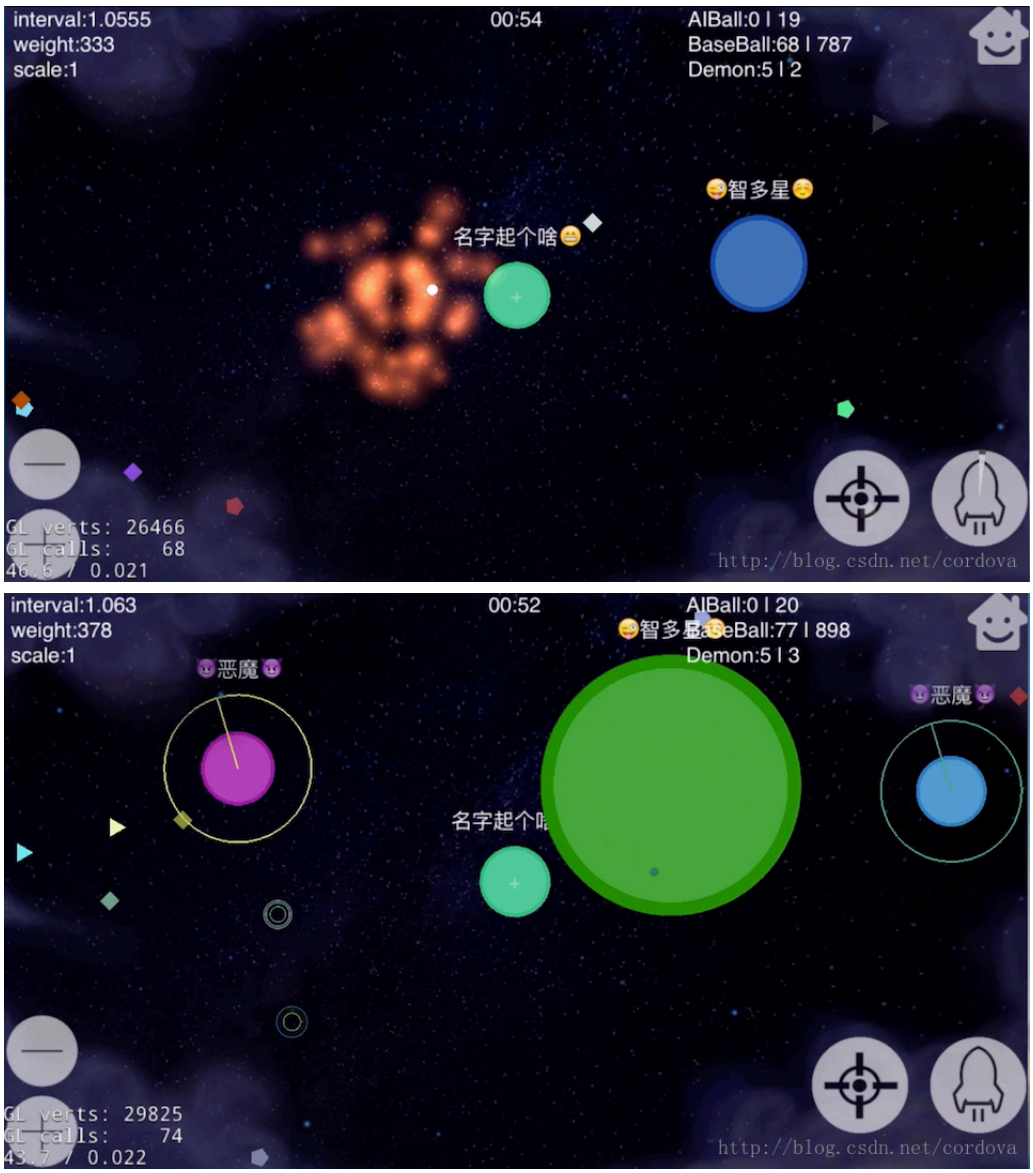
ASTEROID WAR Documentation

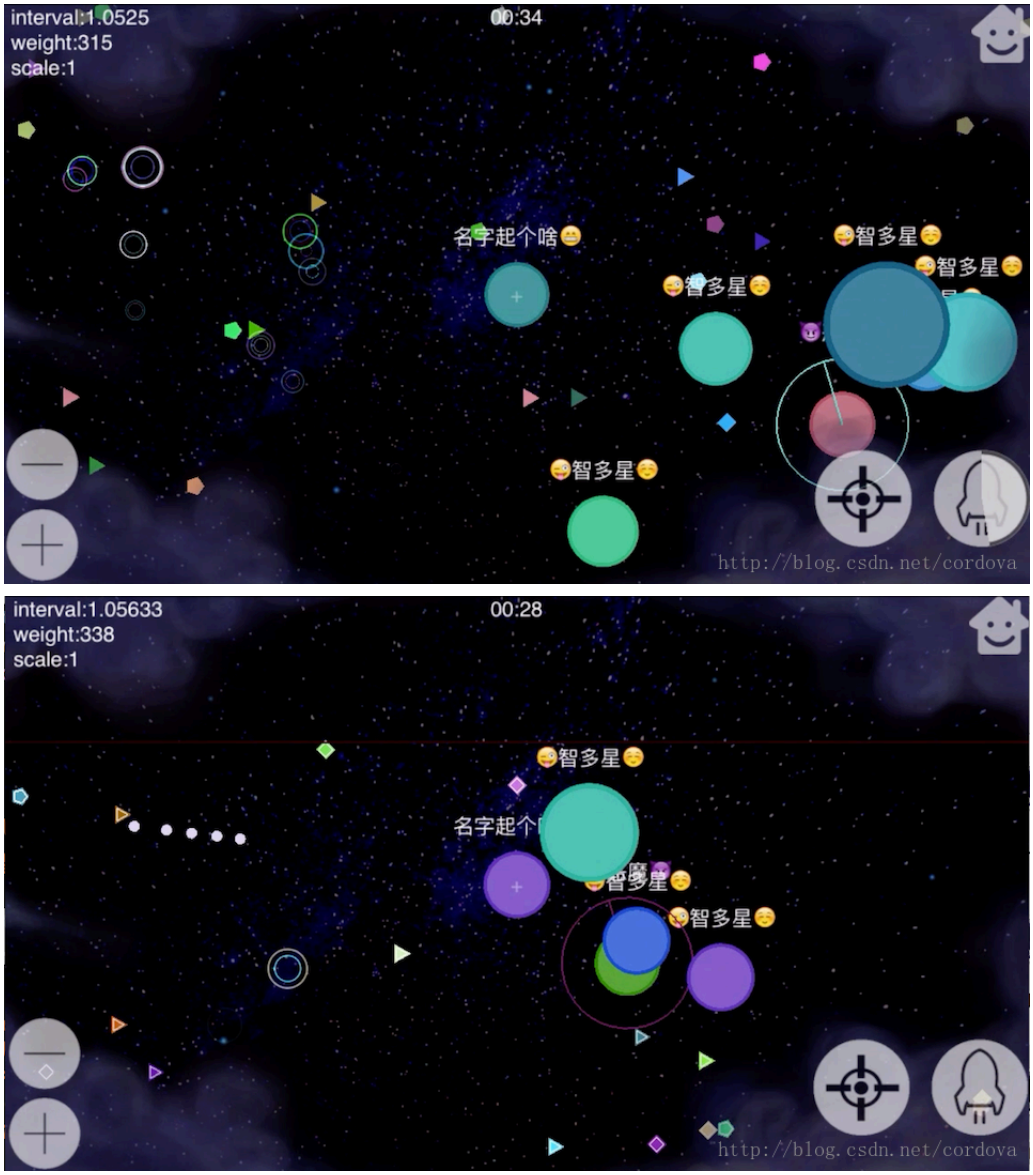
> This is the documentation of ios game ASTEROID WAR. The game is designed by Group T as the group assignment of COMP7506 Smart phone apps development, a course of Hong Kong University in 2016.

Table of contents

- Color Generation
- Game difficulty levels
- User interface design
- Multi-player mode

Preview





Game Preview

1. Group Information

1.1 Member Information

There are four team members in this group. They are:

Name	UID
JIANG, Xinhong	3035347990
PAN, Hao	3035349015
ZHOU, Tingting	3035348621
CHEN, Yingshu	3035349106

1.2 Division of Labor

Common Labor:

- Game design;
- Document writing, video recording;

Individual Labor:

Jiang Xinhou:

- The overall game project construction with cocos2d-x framework on Xcode iOS platform, game development technologies provider;
- Core classes design and implementation;
- Design patterns design and implementation;
- Animation and game effects design and implementation;
- Game algorithm design and implementation, renderring improvement.

Pan Hao:

- Game algorithm design and improvement, algorithm implementation, game improvement algorithm provider;
- Game effect design and implementation;
- Audio effect provider;
- Game test.

Chen Yingshu:

- UI designer and UI resources provider,Game color elements and theme design;
- Partially game improvement algorithms provider;
- Effects and animation of MenuScene design and implementation;
- Game background research.

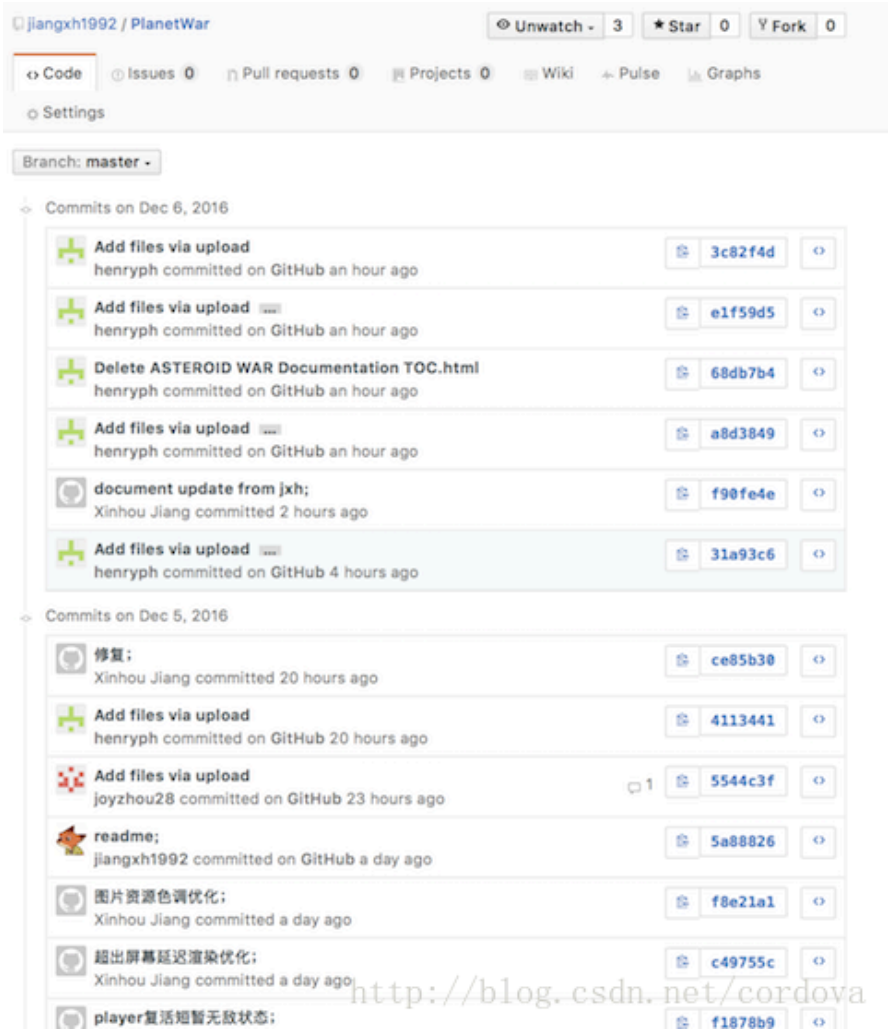
Zhou Tingting:

- Data persistence implementation, user data select and reserve, 'gameover module' implementation;
- Game interaction design and implementation, game events implementation;
- Game test;
- Game background research.

2. Demo Video and Git repository

- A short video demonstrating main features of this game is uploaded to Vimeo at: https://vimeo.com/194161874

- The project version control on Github(commits from Nov 7, 2016 to Dec 6, 2016): https://github.com/jiangxh1992/PlanetWar



3. Background Research

Here list three similar games in the market – Battle of Balls, Nebulous, Agar.io.

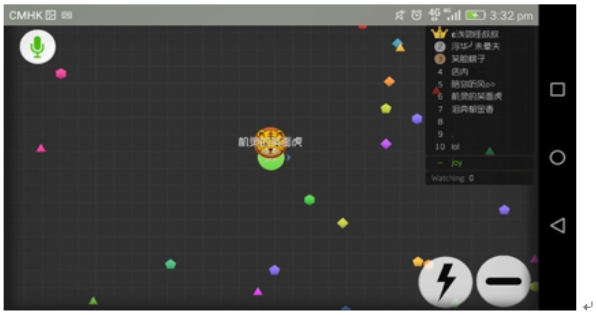
3.1 Battle of Balls

The Battle of Balls is a cutesy but challenging real-time casual game in which you will control a ball to eat other players' balls smaller than yours around you to become the biggest one.

- **Features**:
    - A variety of interactive elements: Players will be able to add friends, follow players, and own their followers, which makes you the stars in the world of Battle of Balls.
    - Spectator Mode: When the big bosses emerge, you can use Spectator mode to watch their fights and root for them
    - Team play: It supports team play together with voice chatting, group chatting.
- **Shortcoming**:
    - A little user-unfriendly, for example, when playing the game, there is no button for pausing or quitting the game; the setting of board of moving area damages the sense of game scene, which really influences user experience.
    - Does not set game difficulty level, you can last the game for very long time unless eaten by others, during which player would easily feel board.
- **Possible improvements**:

  Add some necessary buttons or labels to increase user-friendliness. Set different difficulty levels, for example, in one of levels, the player must obtain some aim weight in limited time, which is considered “pass this level”.

Initial interface



Game playing interface

Game result interface

Ranking interface

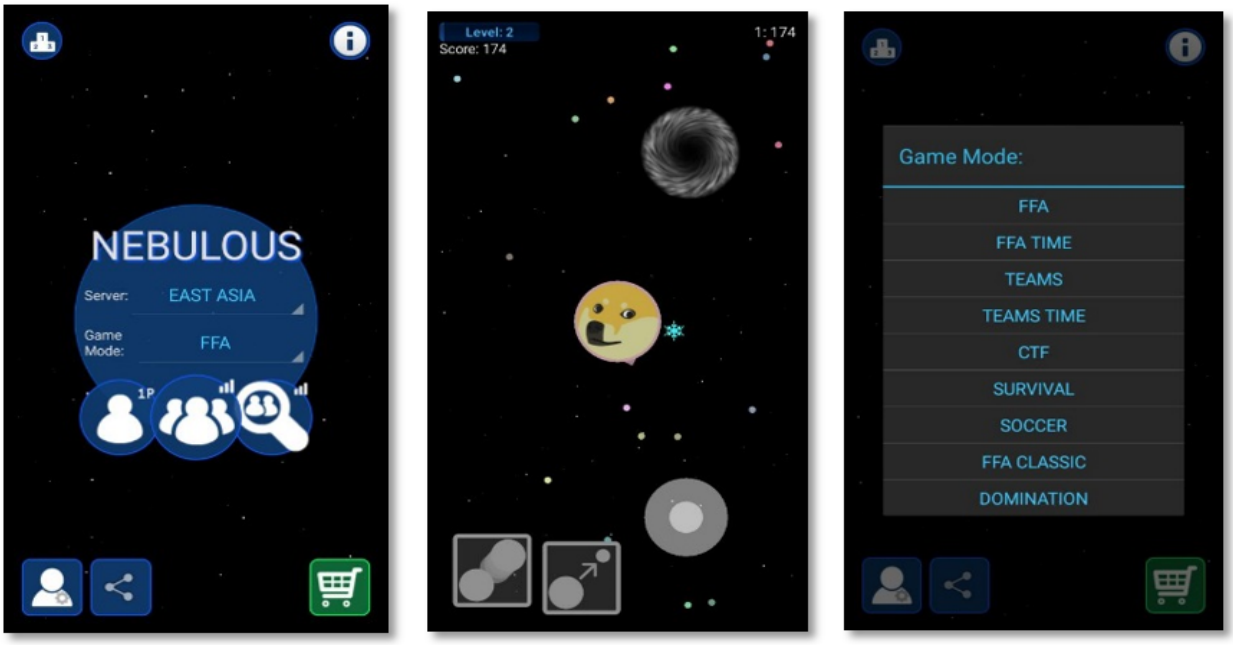
Battle of Balls

3.2 Nebulous

In the Nebulos, you need to grow the blobs by collecting dots placed throughout the game or gobbling up smaller players. Avoid bigger players attempting to do the same. Compete with other players to become the biggest blob.

- **Features**:
  - Players can find groups, join a clan and play with friends.
  - Over 450 skins with unique ways to unlock them and players can upload their own custom skin for other players to see.
  - Online Multiplayer (up to 27 players per game) or Offline Single-player.
  - Various game mode: FFA, Timed FFA, FFA Classic, Teams, Timed Teams, Capture the Flag, Survival, Soccer and Domination Modes!
- **Shortcoming**:

  An evident shortcoming similar to Battle of Balls: the setting of board of moving area damages the sense of game scene, which really influences user experience.



Initial interface **Game playing interface** **Game modes**

Nebulous

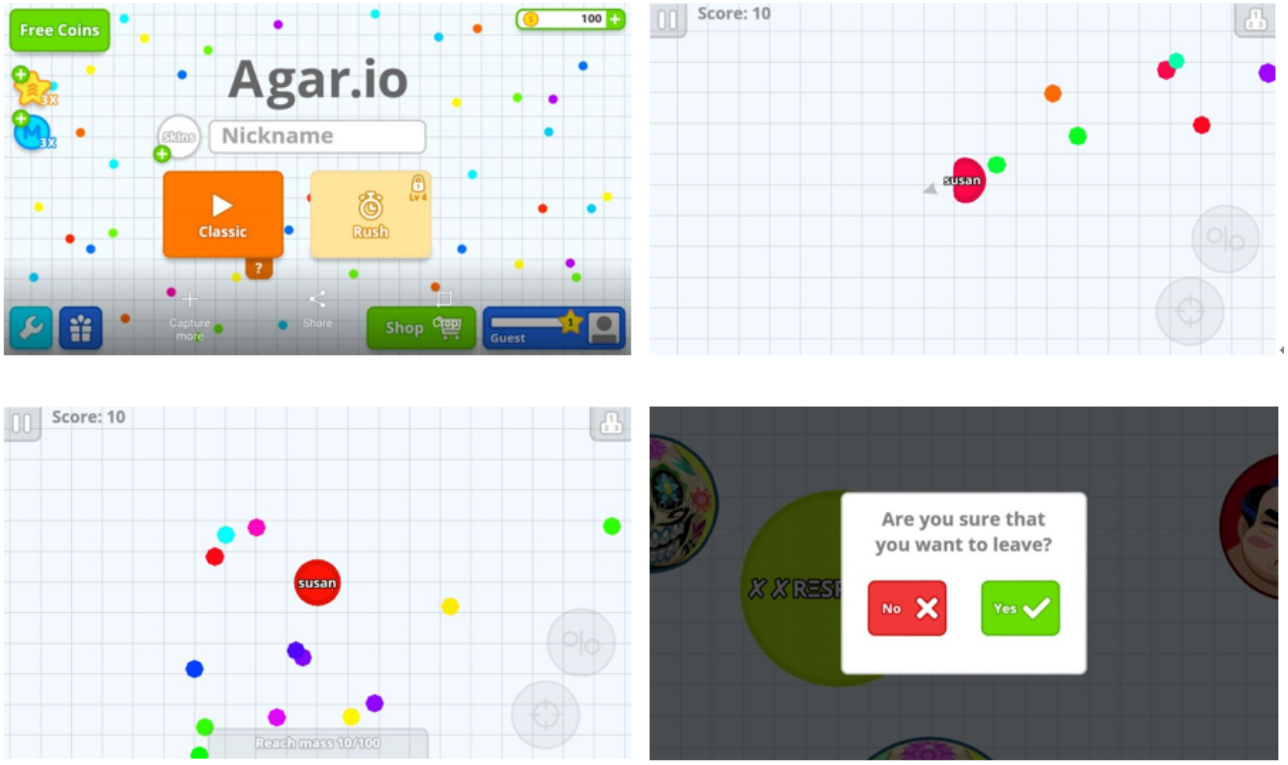
3.3 Agar.io

Agar.io is a multiplayer action game created by Matheus Valadares. The online version of theis game is open-source and is built with socket.IO and HTML5 canvas on top of NodeJS. The mobile version is also available for iOS and Android.

The main features and shortcoming of this game includes:

- **Features**:
  - Online multiplayers
  - A variety of special secret skins with the right username
  - Actions of splitting, shrinking and dodging tactics to catch other players
- **Shortcoming**:

  The game is a bit lagged and there are too many adds.



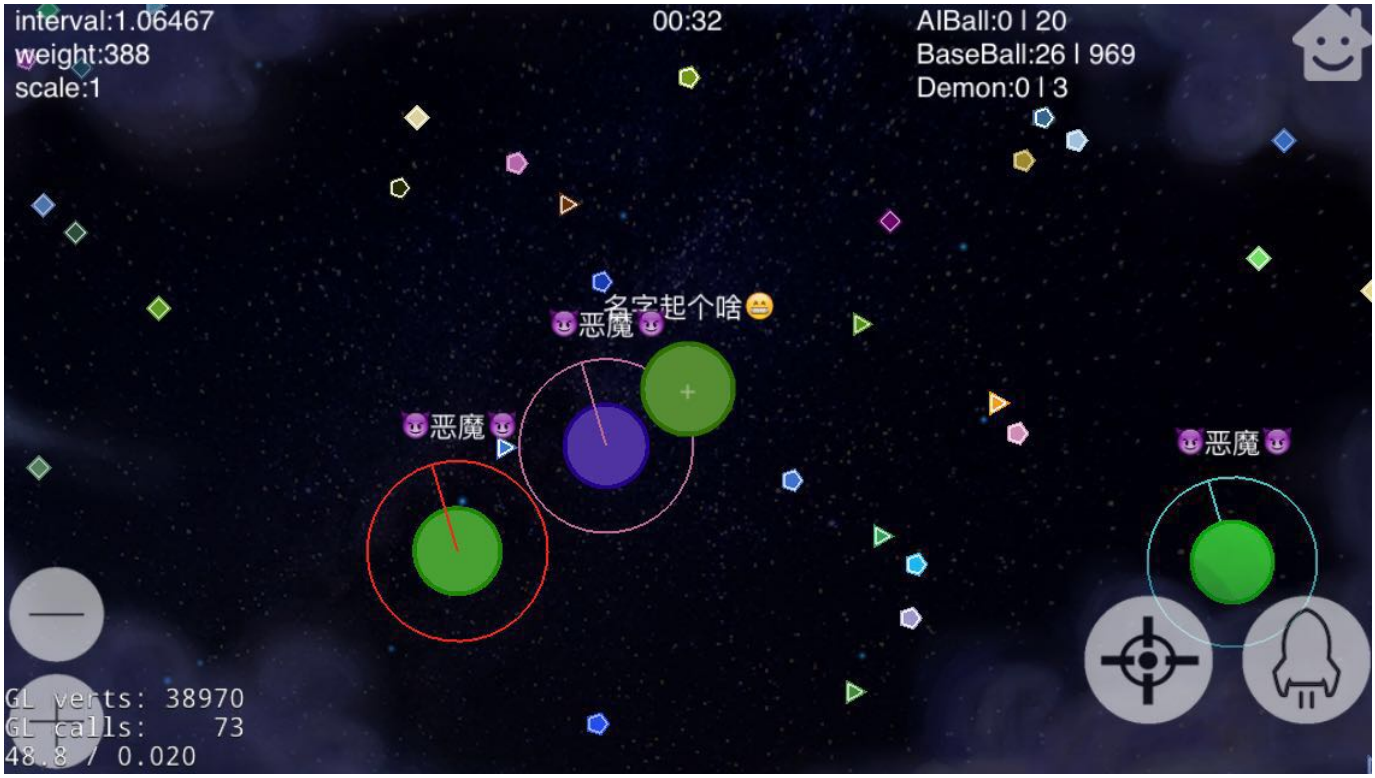
Agar.io

Based on above games, we develop a new game for offline single-player competing with enemies played with artificial intelligence in iOS platform by means of game engine Cocos2d-x. Also, we add more interesting playing modes and skills to our game.

4. Game Overview

It is an entertaining and strategy game application on philosophy of survival of the fittest. To be the strongest species to survive, the asteroid in universe have got to be as much bigger and stronger as possible by eating other smaller ones around it in defense of attacks from enemies.

To be the very surveillance of the universe, keep poking to move, eating others to collect energy and gain weight. It is weight that can proves your ability and power. When your weight is large enough to dominate the universe, you become the king of asteroids in the universe.



game scene

It is your show time to protect your village using your intelligence and skills. The history records will tell you how honorable your achievement is.

5. Detailed Game Description

5.1 Background Story

In Universe, battles never come to the end. Player is one of the bubbles in the village and willing to fight for your home. Player keeps moving to trace enemies and collect energy, player eludes quickly the bigger enemies to sustain your power, until player is powerful enough to protect the village and drive rivals away.

5.2 Objective

The objective is to survive from being encroached and eliminate enemies. To be powerful enough, the hero is supposed to eat smaller ones as many as possible to gain weight and get a larger size. The hero also can use bullets to defeat demons to get higher score. The player can break the history record by eating lots of asteroids and defeating as many demons as possible. To add the challenge of the game, we set two game modes – unlimited-time and timer.

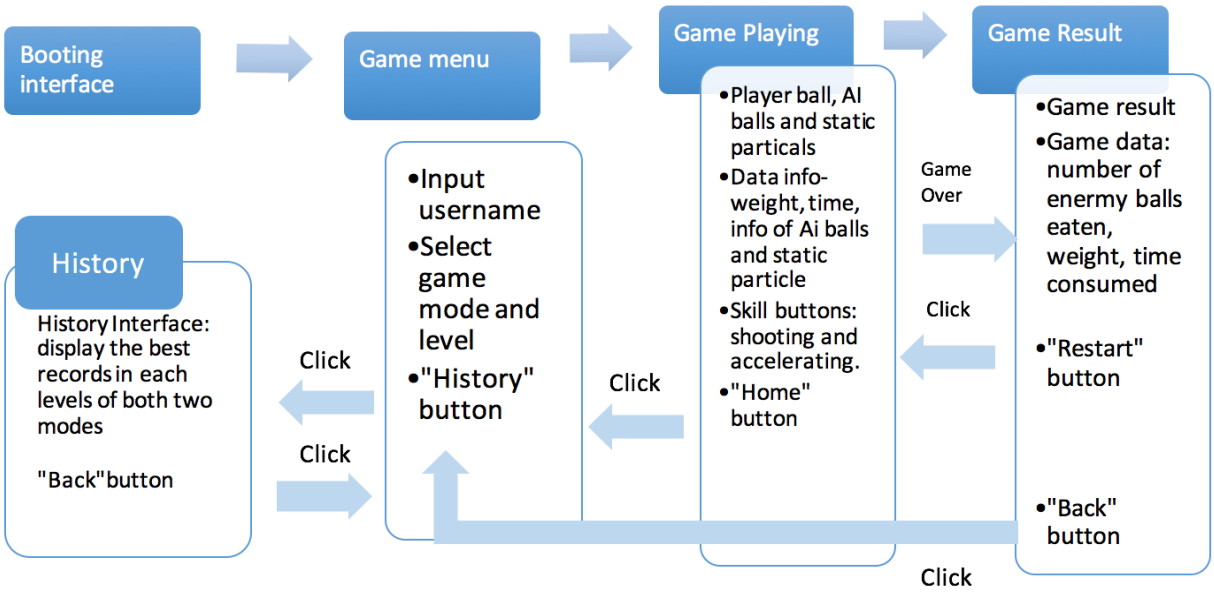
In the unlimited-time mode, the player has to control the hero to eat other asteroids and defeat demons as many as possible. Meanwhile, the hero should avoid being encroached or attacked by the bigger ones,

otherwise the game will be over. The final score will be calculated when the game is over and the highest score will be recorded.

In the timer mode, the hero tries to eat smaller ones and defeat demons as many as possible within a period of certain time. The final score will be calculated when the limited time is up or the hero dies in halfway and the highest score will be recorded.

5.3 Gameplay

5.3.1 Storyboard



storyboard

1) Booting interface

There is an unavoidable welcome interface showing a short period of time on the screen, displaying the game title, main characters and content.

Booting interface

2) Menu interface

The menu is the interface consisting main scene with animated background and several UI components, used by the player to input username, select game mode and difficulty level. Player can select “limited time” or “unlimited time” game mode, and select game levels. The initial game level is Level 1 and player has to pass the last level to unlock the next level. After inputting above info, once clicking the “Strat” button, the player will enter the game playing scene.

There is a “History” button used to skip to game history interface, which would display best game records of different game levels.

Menu interface

3) Game interface

After clicking “Start” button in the Menu interface, the game starts and the hero appears at the center of the screen.

The background is a galaxy universe and the background will change correspondingly as the hero moves. The dynamic rival asteroids (called AI asteroids and demon asteroids) controlled by AI and static tiny particles with a certain weight are allocated here and there in the universe. The hero moving by oriented moving gestures of the player can eat smaller rivals and any tiny particles. And he rivals controlled by artificial intelligence would also do that.

When asteroids eat others, their weights and sizes get larger, but the moving speed would be reduced. That is, the moving speed is inversely proportional to the weight while the size (radius) of the asteroid is proportional to the weight.

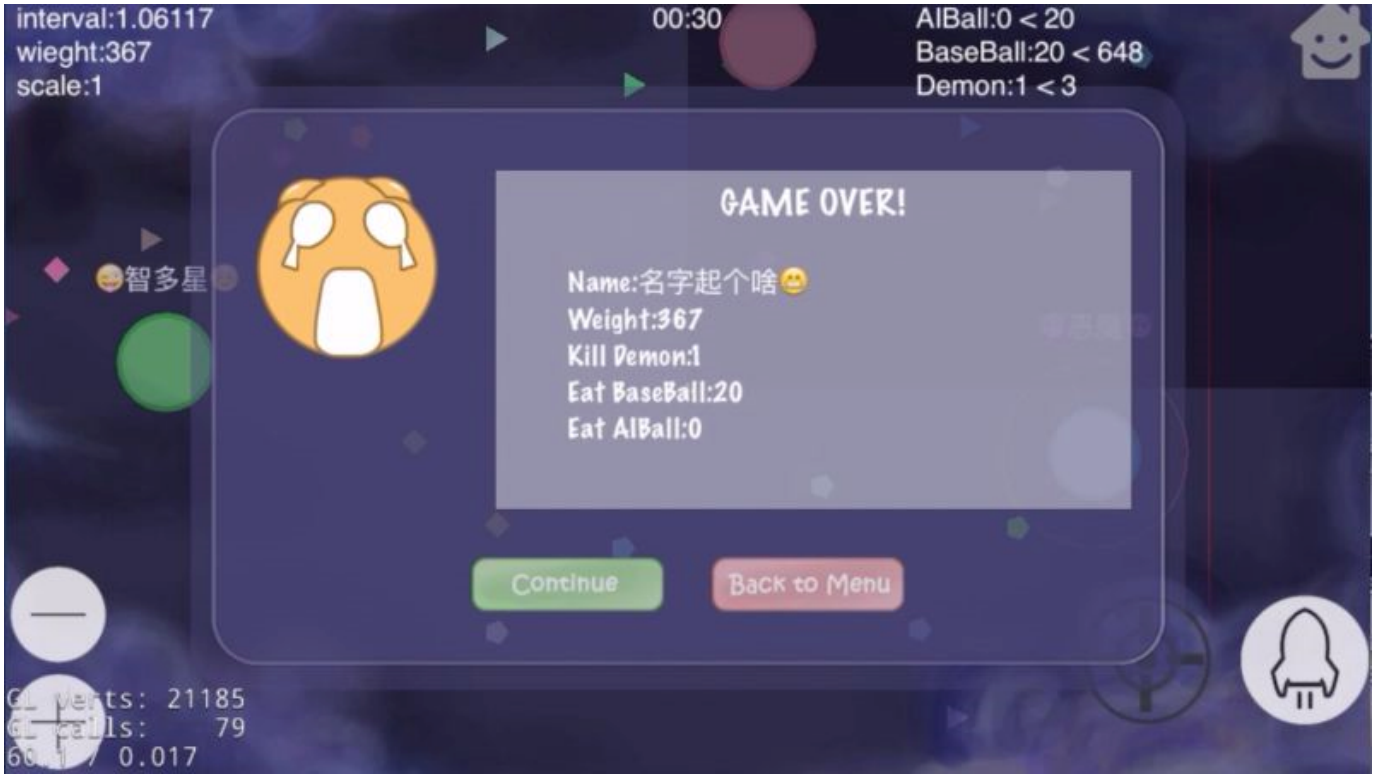
Here comes to a special case that hero cannot eat demon asteroids but can only shoot tiny particles to them to decrease their weights. When playing, the player can click the buttons on the bottom right of the interface -- “Shooting” and “Dash”. One is used to attack the demon asteroids, and the other is a special and necessary skill for the hero. When the size of hero is very large, there is no denying that the moving speed slows down and it is hard to catch the smaller ones. So hero can adopt such a strategy -- dividing itself into several asteroids to speed up moving and get more chance to eat enemies. Certainly, hero would also take more risks of being eaten by others. So considering which situation the hero should be split or merged adds challenges to this game.

The final score will be calculated according to the number of asteroids the hero ate and defeated when the game is over or time is up, and the highest score will be recorded.

Game interface and UI components

4) Game result interface

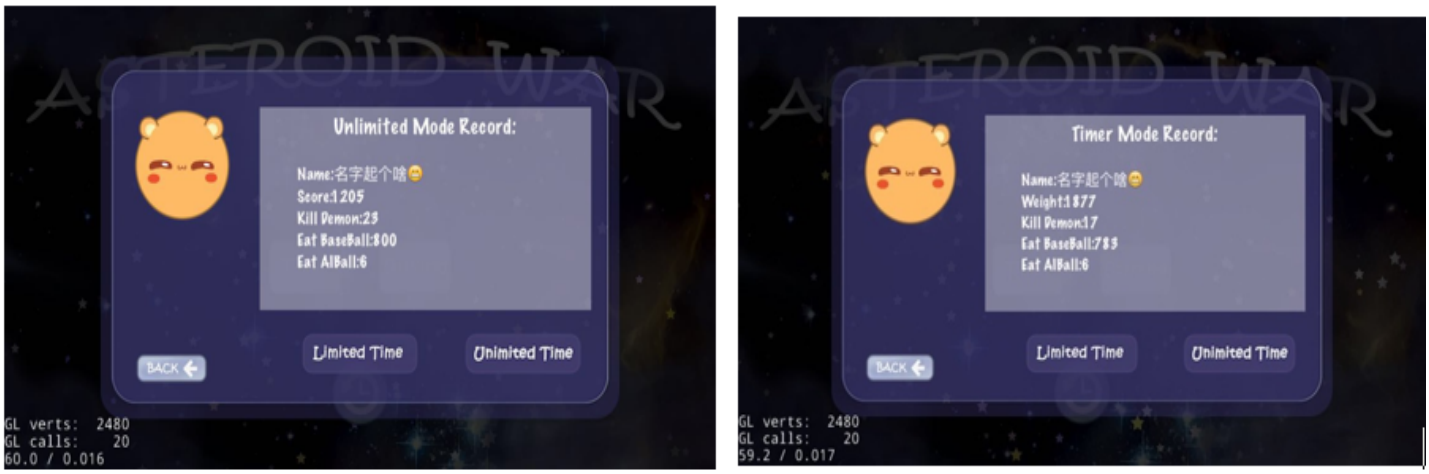
When the game is over, game result information will pop up consisting of data of weight, time, number of rival asteroids eaten by hero. There are buttons for restarting the game and returning the menu interface.



Result interface

5) History interface

There is a table showing best game records of different levels and a button to return the menu interface. On the menu interface, the player selects the game mode and level.



Game records for Unlimited mode and Timer mode

5.3.2 Game Mode & Game Levels

On the menu interface, the player selects the game mode and level.

There are two game modes -- unlimited-time mode and timer mode. Levels are set from easiest to hardest for each game mode. Accordingly, the game history will record the best record of each level. In the initial state, only Level 1 is unlocked and only when the last level is passed will the following levels be unlocked. Otherwise the player cannot enter the next level. Each level is set with different prescribed playing time, target weight and AI asteroids settings. As the level of difficulties increases, target weight will be larger (in unlimited mode), the number of demons and AI asteroids will be larger, and the demons will become more sensitive as well.

In our prototype version, only game modes are taken into account. As the time passed by, the demons will be more sensitive to the hero and it will be harder to get higher score.

5.3.3 The core of Gameplay

The key point of gameplay is that the single player is supposed to annihilate as many smaller asteroids as possible and avoiding being eaten. The moving speed is inversely proportional to the weight while the size (radius) of the asteroid is proportional to the weight. The shooting and speeding up skills are essential for the player to strategically utilize in order to gain a great score in the game.

5.3.4 Game Difficulties

Different game levels or game difficulties lies on prescribed game time (if it is limited), number of enemies (AI asteroids), sensibility of demon asteroids, aimed weight and initial weight of AI asteroids. All these factors should be taken into consideration to break a new record of the game, flexibly using skills speeding up or shooting to escape the tracing from enemies and annihilate more other asteroids.

5.4 Characters

There are four types of characters in this game:

- Hero: it moves and uses skills by swiping and clicking the buttons, controlled by the player. This is realized by class PlayerBall in the code.
- Tiny static particles: they spread around the background and can be eaten by dynamic asteroids providing basic energy. This is realized by class StaticBall in the code.
- Enemy asteroids controlled by artificial intelligence: AI asteroids and demon asteroids. They move automatically getting close to the nearest asteroids. There is an algorithm to calculate the moving direction and speed with conditions the asteroid weight and the distances of surrounded asteroids. There is difference between AI asteroids and demon asteroids, the player cannot directly eat the demon asteroid but can attack them by meaning of shooting tiny asteroids to them. These are realized by class AIBall and Demon in the code.

5.5 Controls

In the menu page, the player can select different game levels and modes. And start a new game by clicking the “Start” button. The game history can be viewed by clicking the “History” button.

In the history page, we can view the game records and the “Back” button is to be clicked to return the menu page.

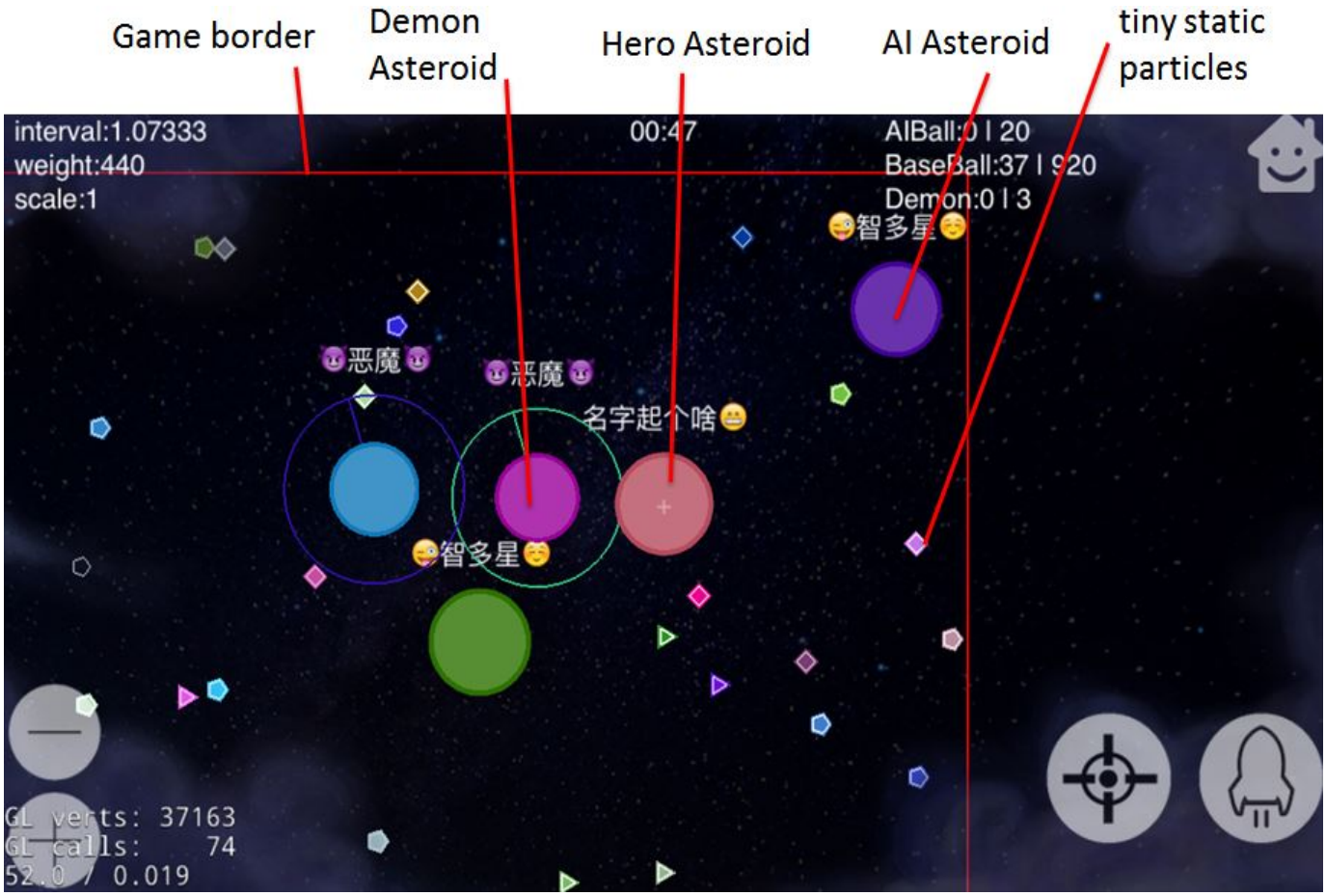
In the game page, the player can control the hero by swiping the screen indicating the moving direction and tap the skill buttons “shooting” and “dash” on the bottom right. The “Home” button is on the top right, which can be clicked to quit the game.

5.6 Graphics

All graphics used in the application is of fantasy and cool style. All background images are with dark hue similar to galaxy and universe. Buttons look like bubbles and all characters are designed in round shape in cute style. The whole design is user-friendly.

There are four characters, the hero, AI asteroids, demon asteroids and tiny static particles. AI asteroids are of normal bubble shapes in different colors with names on the top of asteroid figures. Demon asteroids

have different style with common AI asteroids – they are with sparkling halos. The plenty of static particles are in different colors and shapes randomly.



Characters and their graphic display

5.7 Sound and Music

Sounds and background music are selected in delightful and funny manner with a little exciting sense.

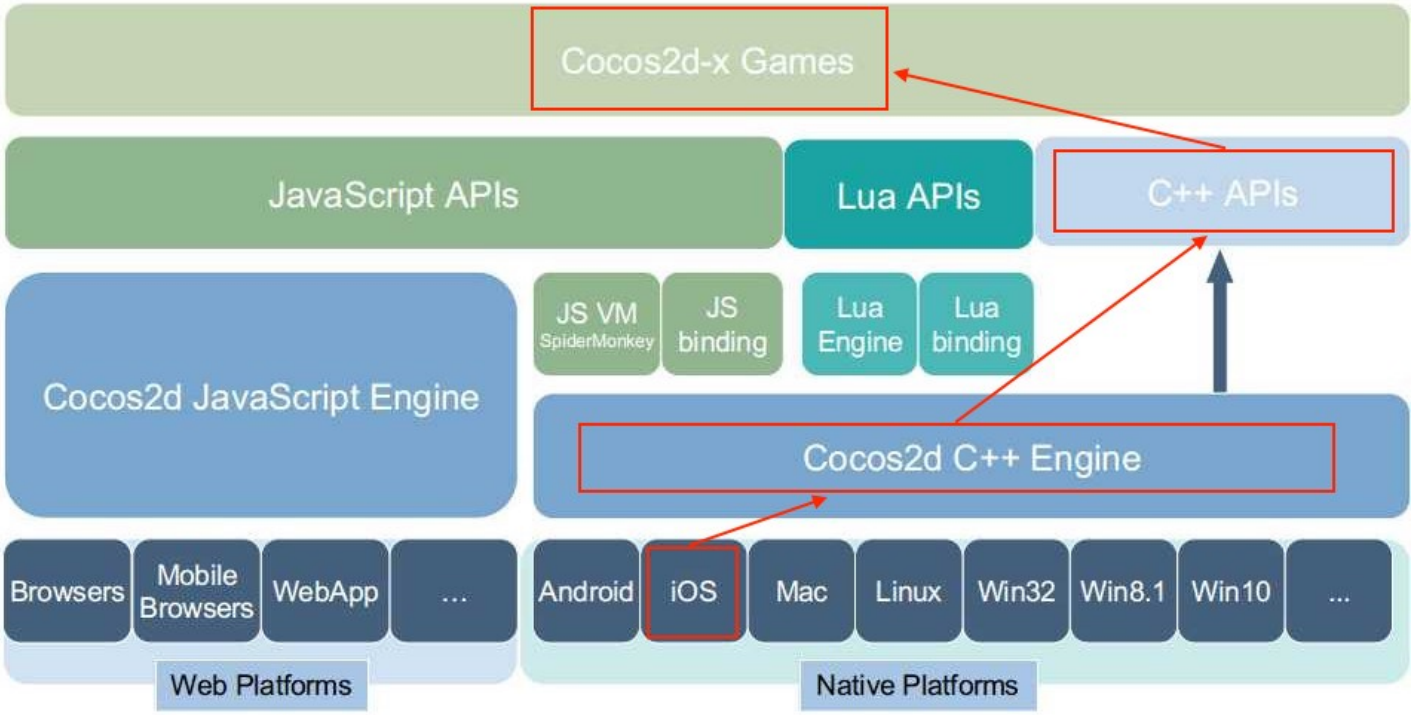
Menu and history pages, and main game page apply 2 different pieces of music. Former one is more delightful while later is more competitive and funny. All buttons have the same sound effect. As for the game result pages has two sound effects applied for two results, a new best record or a normal score.

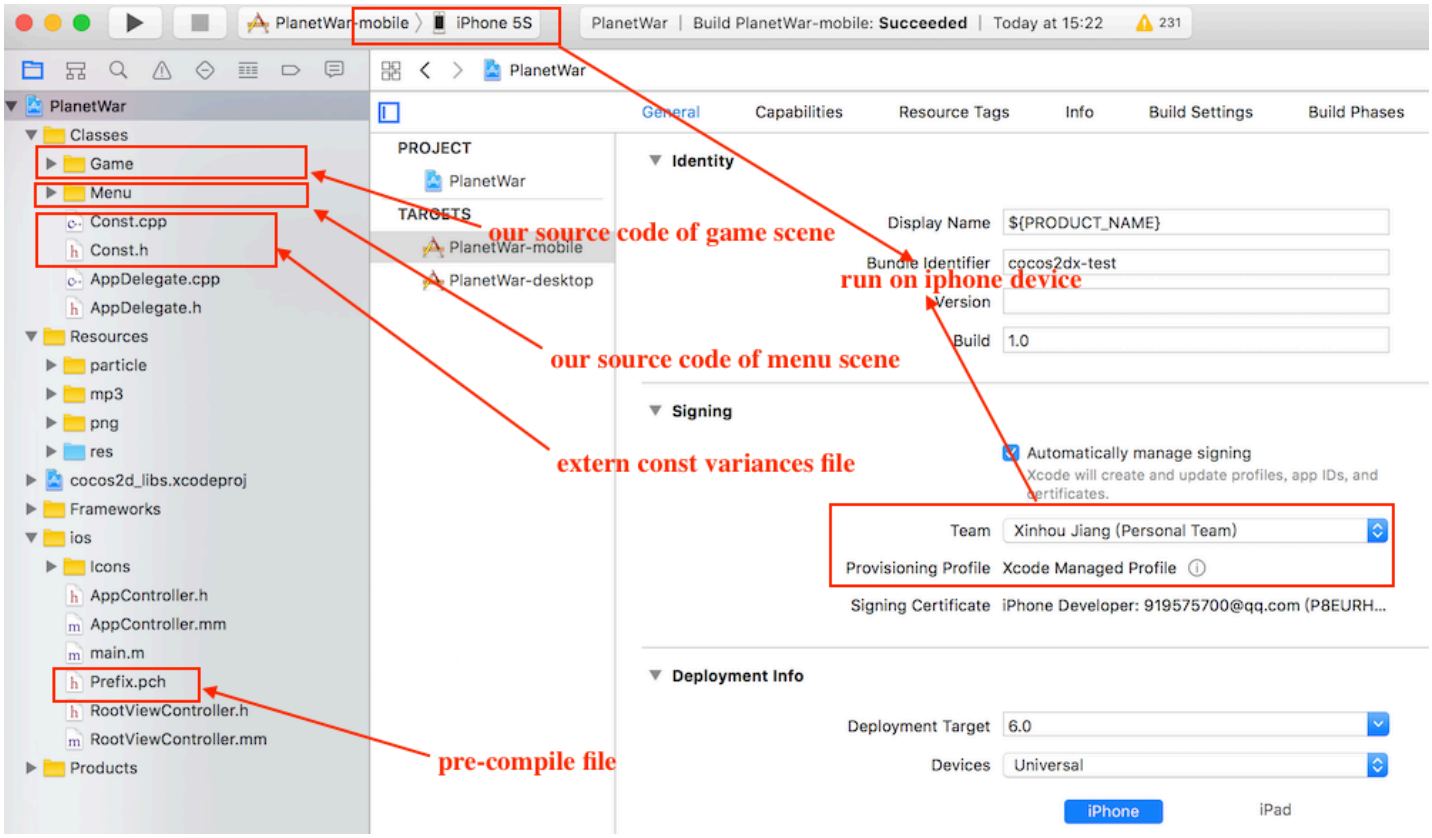
6. Programming implementation

6.1 About cocos game engine

Cocos is one of the most popular 2d game engine and the original version is cocos2d-iphone, which is specially used for iphone game development with objective-c. Cocos2d-x is a opensource and cross-

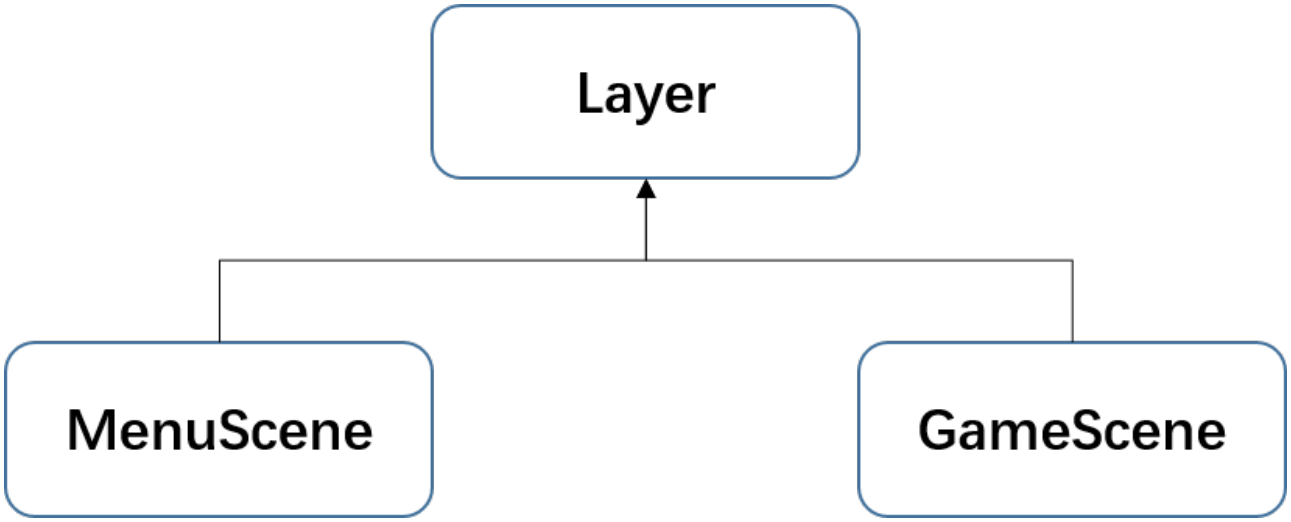
platform version of that. By using Cocos2d-x, developer can easily develop 2d game and even 3d game now with C++ or javascript and can easily plant the game to almost all the platforms like iOS, Android, WP and PC platforms(MAC, Windows, Linux etc).The framework of the cocos2d-x game engine can be seen form the picture below, and here we develop our game on iOS platform with C++. In this app, we will use several class and methods of Cocos2d-x.





6.2 Layer

In Cocos2dx, `Layer` is a subclass of `Node` that can receive touch events and contain any `Node` as a child, including `Sprites` and other Layer objects. In this game, the inheritance of `Layer` is as follows:



Layer in this game

We create two layers `MenuScene` and `GameScene` that inheritate from `Layer` and use function `createScene()` to return the scenes that contain the new created layer. Take `MenuScene` as example, we create this layer as following:

```
class MenuScene : Layer {
public:
    // 创建对象
    static Scene* createScene();
    // 对象初始化
    virtual bool init();
    // 安帧更新
    virtual void update(float time);
    CREATE_FUNC(MenuScene);
    // 场景退出
    void onExit();

    float scaleCount = 0;

    // 动态UI引用
    Sprite *label_title;
    MenuItemImage *item_startgame1;
    MenuItemImage *item_startgame2;
    MenuItemImage *item_history;
    Sprite *menu_bg;
    LayerColor *history_layer_bg;

    // History UI引用
    Label *label_history;
    Label *label_name;
    Label *label_weight;
    Label *label_demon;
    Label *label_base;
    Label *label_ai;

    // 添加UI
    void addUI();
    void addHistoryUI();

    // 事件函数
    void startGameTimer();
    void startGameUnlimited();
    void openHistory();
    void showLimitedRecord();
    void showUnLimitedRecord();
    void hideHistory();
    void updateHistory(GAME_TYPE type, __String new_name, int new_weight, int new_
baseball, int new_aiball, int new_demon);
};
```

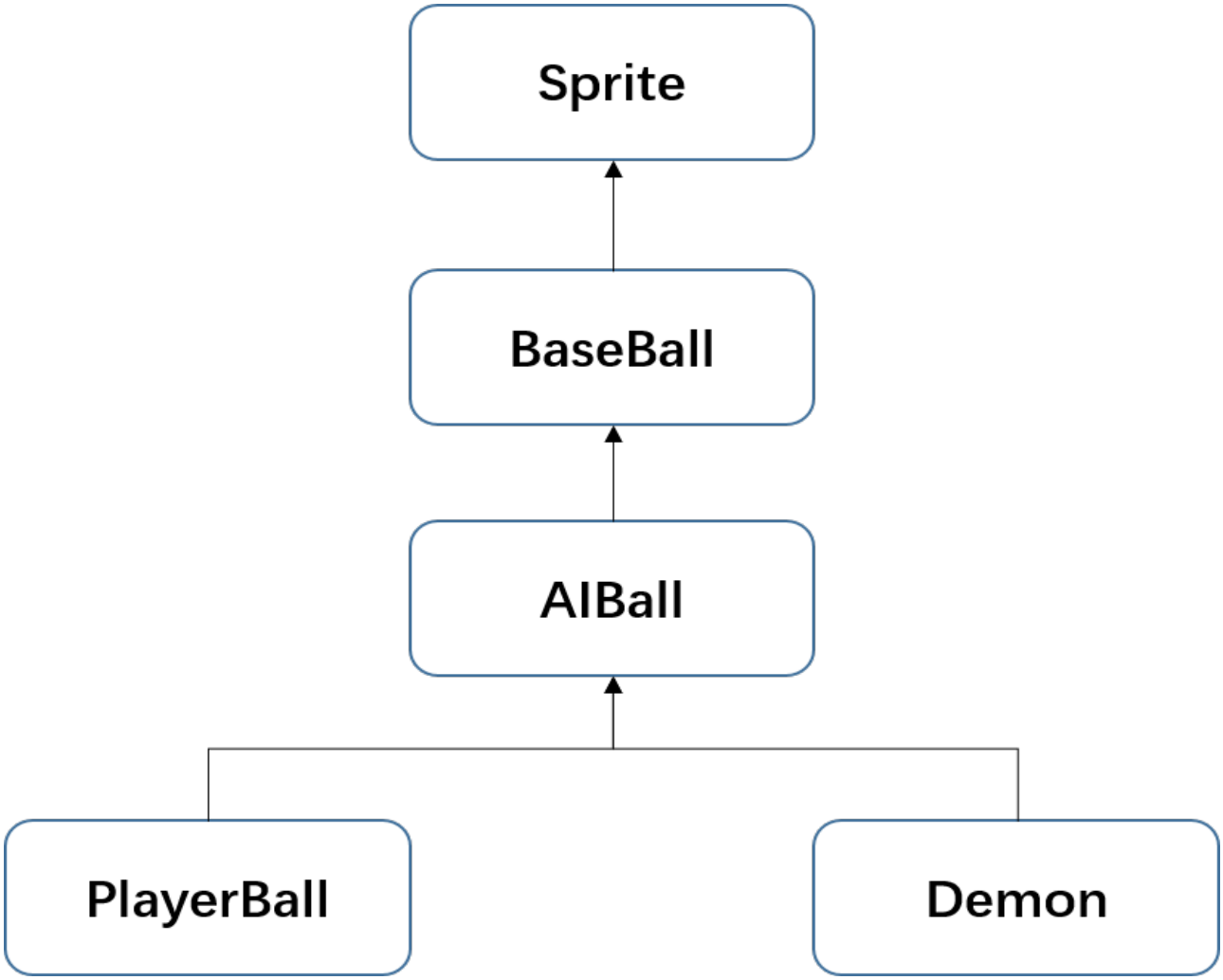
Then we use function `createScene()` to return the scene that contains this layer:

```
Scene* MenuScene::createScene() {
    auto scene = Scene::create();
    auto layer = MenuScene::create();
    // add layer as a child to scene
    scene->addChild(layer);
    return scene;
}
```

We use `MenuScene` for the Menu interface and `GameScene` for the Game interface.

6.3 Sprite

In Cocos2dx, `Sprite` is a subclass of `Node` that can be moved, rotated, scaled, animated, and undergo other transformations. In this game, the inheritance of `Sprite` is as follows:



Sprite in this game

6.3.1 Baseball

`Baseball` is a virtual base class of all ball instances that can move and change size including playerBall, AIBall and demon. This class contains variables including position, radius, color, weight of the target ball, a `drawNode` method and a set of getters and setters.

```
class BaseBall : public Sprite {

/** 重写函数 **/
public:

/** 内部变量和函数 **/
protected:
    Vec2 position = Vec2::ZERO;         // 坐标
    double radius = 0;                  // 半径
    Color4F color = Color4F::YELLOW;    // 颜色
    int weight = 1;                     // 重量
    bool isDraw = true;                 // 是否启动图形绘制
    DrawNode *drawNode = NULL;          // DrawNode

/** 对外接口 **/
public:
    // getter
    const bool getIsDraw()const{return isDraw;}
    const Vec2 getPos()const{return position;}
    const float getR()const{return radius;}
    const int getBallWeight()const{return weight;}
    const Color4F getBallColor()const{return color;}

    // setter
    void setIsDraw(bool isdraw){isDraw = isdraw;}

    // 缩放
    virtual void scale(const float scale){}
};
```

6.3.2 AIBall

`AIBall` is a subclass of `BaseBall` and the base class of `PlayerBall` and `Demon`. The instances of this class have the following features:

- They can eat the static ball and other AI ball instances that are smaller than themselves, including the player ball that is controlled by the player.
- They will move automatically and randomly. If the player ball is in their detectable area, they will chase the player ball if they are bigger than it and will get away from the player ball if they are smaller than it. This is realized by the method `gameObserver` in `Game.cpp` as follows:

```
        for (Vector<AIBall*>::const_iterator it = AIBallArray.begin(); it != AIBal
lArray.end(); it++) {
        AIBall *aiball = *it;
        AIBall *player = Game::sharedGame()->player;
        Point p = player->getPos();
        int weight = player->getBallWeight();
        // squared distance of centers of aiball and player ball
        float distance2 = pow(p.x - aiball->getPos().x, 2.0) + pow(p.y - aiball->g
etPos().y, 2.0);
        // distance of aiball and player ball
        float distance = sqrt(distance2) - player->getR() - aiball->getR();
        if (distance < ScreenHeight) {

            Vec2 dir = aiball->getPos() - p;

            if (aiball->getBallWeight() > weight) {
                dir = -dir + aiball->getDirection(); // chase
            }else {
                dir += aiball->getDirection(); // flee
            }

            dir.normalize();
            aiball->setDirection(dir);
        }
    }
```

- When the radius of a AI ball instance is larger than a quarter of the screen height, this ball will 'explode' and this object will be removed.
- The AI balls will be created periodically on random positions according to the maximum number of demons and the existing number of demons in the game.

6.3.3 PlayerBall

The `PlayerBall` is a subclass of `AIBall` and the instance of this class is the ball that controlled by the player. The player ball instance has the following features:

- It can eat the static ball and all AI ball instances that are smaller than itself.
- Listen touch events. When a drag event is triggered, the start point and end point of the touch event are formed a vector served as the moving direction of the player ball. If the it is a touch event only and the start point and end point of this event is too close, the player ball will be static. This is realized by the method `addTouchListener` in `Game.cpp`
- When the button of **shoot** is clicked, the `shoot` method in `Game.cpp` will be called to realize the effect that the player shoot 'bullets' in the direction of itself. The 'bullets' are used to attack the 'demons' to reduce their size and destroy them.
- When the button of **dash** is clicked, the `dash` method in `Game.cpp` will be called to realize the

speed-up effect of the player ball. The dash button will be disabled after clicked and resumed after 500 frames.

6.3.4 Demon

The `Demon` is a subclass of `AIBall` and the instances of this class have the following features:

- They can eat the static ball to increase size but cannot eat a smaller AI ball as the other AI ball or player ball do.
- When they collide with the player ball, the weight of player ball will be lost constantly until two balls are separate. When thay collide with the AI bal, they will be eaten if the AI ball is bigger.
- They will lose weight (reduce size) when they are 'attacked' by the bullets that are shooted from the player ball when the button of **shoot** is clicked. When their weight is reduced lower than 50, it will be killed.
- They will move automatically and randomly. If the player ball is in thier detactable area, they will chase the player ball no matter what their size are. Similarly as the AIBall, this is realized by the method `gameObserver` in `Game.cpp` as follows:

```
 for (Vector<Demon*>::const_iterator it = DemonArray.begin(); it != DemonArray.end
(); it++) {
        Demon *demon = *it;
        Vec2 newDir = player->getPos() - demon->getPos();
        if(newDir.x*newDir.x + newDir.y*newDir.y > ScreenWidth*ScreenWidth*4) cont
inue;
        newDir.normalize();
        demon->setDirection(newDir);
    }
```

- The demon instances will be created periodically on random positions according to the maximum number of demons and the existing number of demons in the game.

6.4 StaticBall and Bullet

Unlike the ball class above, the `StaticBall` and `bullet` are just data structure but not classes that inheritated from the Sprite since transformations on these objects are not needed. By doing this, the production of significant number of instances is avoided and these objects are created by the `drawNode` method (`drawPolygon` for `StaticBall` and `drawDot` for `bullet`). For example, the data structure of `StaticBall` is as follows:

```
class StaticBall {
public:
    bool isActive;                    // 是否存活
    Vec2 position = Vec2::ZERO;       // 坐标
    float radius = 0;                 // 半径
    int polyNum = 3;                  // 边数
    Point *vertexs;                   // 顶点数组
    Color4F color = Color4F::YELLOW; // 颜色
    int weight = 5;                   // 重量

    // 构造
    StaticBall();
    // 重新激活
    void reActive();
    // 缩放
    void scale(float scale);
    // 析构
    ~StaticBall();
};
```

The `StaticBall` objects have the following features:

- They have fixed size, weight, color and position after being created. They are static and will be destroyed when they collide with all other AI balls, player balls and demons.
- Random number of static balls will be created periodically on random positions according to the maximum number of static balls and the existing number of static balls in the game.

The `bullet` objects have the following features:

- They have fixed size, color, speed and direction after being created. They will move in the directon of player ball.
- They are used to reduce the weight of the demons and will be created when the button of shoot is clicked by the user.

6.5 Scaledown and Scaleup

There are a button of Scaledown and a button of Scaleup to realize the effects of zooming the camera in the game scene. These two effects are triggered when the buttons are touched and the method `scaleScreen` in `Game.cpp` is called.

The game is scaled by:

```
  //maxW and maxH are the width and height of the game border
  maxW *= scale;
  maxH *= scale;
```

The ball objects (including AIBall, PlayerBall, demon and StaticBall) are scaled by scaling their positions, radius, speed (no speed for the StaticBall).

```
// postion
position *= scale;
// speed
speedInterval /= scale;
// radius
radius = sqrt(weight*scale);
```

However, the label of ball objects are not included in the scalling.

6.6 Draw Order

In our game, we determines which node is drawn behind or in front of another node by setting their `zOrder property. The node with the smallest zOrder property is drawn first and so will be placed at the bottom. The zOrder of different labels, sprites and layers in our game are as follows:

- Background: -100001
- Static Ball: -100000
- Bullet: -100000
- Uilayer and all buttons on Game Scene: 100000
- AIBall, PlayerBall, Demon: from -100000 to 0, larger ball has larger zOrder. This is realized by the following,

```
drawNode->setGlobalZOrder(radius-100000);
```

By doing this, the background will be placed so that every other objects are above it and the buttons are placed at the highest layer so that will not be coverd by the balls. The bigger ball will have larger zOrder so that it will always on the top of the smaller ball.

6.7 Data persistence

After a game is ended, the final weight of the player ball will be compared with the stored data of final weight and keep the game record of the game that has a larger weight. The game record data will be stored for two different game modes and the this process will be realized the `updateData` method in the `Gamp.cpp`.

```
bool Game::updateData(string name, int new_weight, int new_baseball, int new_aibal
l, int new_demon) {
    if (gameType == GAME_TIMER) {
        // 取出旧数据
        int weight = UserDefault::getInstance()->getIntegerForKey("timer_weight",
-1);
```

```
        //int demon = UserDefault::getInstance()->getIntegerForKey("timer_demon",
-1);
        //int baseball = UserDefault::getInstance()->getIntegerForKey("timer_baseb
all",-1);
        //int aiball = UserDefault::getInstance()->getIntegerForKey("timer_aiball"
,-1);
        // 存入新记录数据
        if (new_weight > weight) {
            UserDefault::getInstance()->setStringForKey("timer_name", name);
            UserDefault::getInstance()->setIntegerForKey("timer_weight", new_weigh
t);
            UserDefault::getInstance()->setIntegerForKey("timer_demon", new_demon)
;
            UserDefault::getInstance()->setIntegerForKey("timer_baseball", new_bas
eball);
            UserDefault::getInstance()->setIntegerForKey("timer_aiball", new_aibal
l);
            return true;
        }
        return false;
    }
    else if (gameType == GAME_UNLIMITED) {
        // 取出旧数据
        //int weight = UserDefault::getInstance()->getIntegerForKey("unlimited_wei
ght", -1);
        int demon = UserDefault::getInstance()->getIntegerForKey("unlimited_demon"
, -1);
        int baseball = UserDefault::getInstance()->getIntegerForKey("unlimited_bas
eball",-1);
        int aiball = UserDefault::getInstance()->getIntegerForKey("unlimited_aibal
l",-1);
        // 成绩加权
        int power = baseball + aiball*10 + demon*15;
        int new_power = new_baseball + new_aiball*10 + demon*15;
        // 存入新记录数据
        if (new_power > power) {
            UserDefault::getInstance()->setStringForKey("unlimited_name", name);
            UserDefault::getInstance()->setIntegerForKey("unlimited_weight", new_w
eight);
            UserDefault::getInstance()->setIntegerForKey("unlimited_demon", new_de
mon);
            UserDefault::getInstance()->setIntegerForKey("unlimited_baseball", new
_baseball);
            UserDefault::getInstance()->setIntegerForKey("unlimited_aiball", new_a
iball);
            return true;
        }
        return false;
```

```
    }
}
```

7. Optimization Algorithm

7.1 Rendering

In this game, we will only render the objects including AI balls, demons and static balls that are within the screen to improve the fps of the game. Take the `draw` method of `AIBall` as example and code is following:

```
void AIBall::draw(cocos2d::Renderer *renderer, const cocos2d::Mat4 &transform, uin
t32_t flags) {

    // 超出屏幕不渲染 Not render balls that are out of the screen
    Vec2 playerP = Game::sharedGame()->getPlayer()->getPos();
    if(abs(position.x - playerP.x) > (ScreenWidth/2+radius) || abs(position.y - pl
ayerP.y) > (ScreenHeight/2+radius)) return;

    // 清空之前的绘制
    drawNode->clear();
    if (isDraw) {
        // 绘制实心圆形
        drawNode->drawDot(Vec2(0, 0), radius, color);
        drawNode->drawDot(Vec2(0, 0), radius*0.9, Color4F(1.0, 1.0, 1.0, 0.2));
        drawNode->drawCircle(Vec2(0, 0), radius, 360, radius, false, Color4F(color
.r, color.g, color.b, 0.5));
        // 根据球的半径更新当前球的绘制深度，半径越大的绘制在前面覆盖更小的球
        drawNode->setGlobalZOrder(radius-100000);
    }
}
```

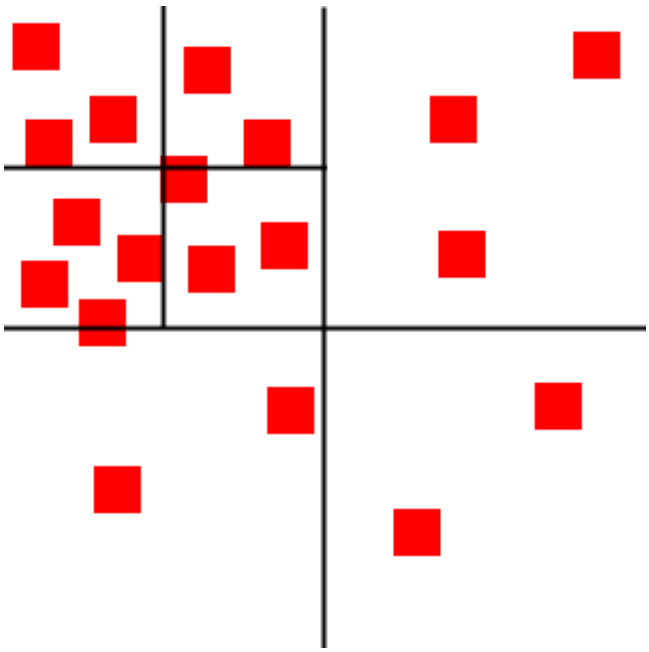
7.2 Quadtree

In this application, the collision detection will be conducted by checking all objects stored in a 1-D array. For example, for the player ball only, it need to check all static ball objects and all AI ball objects for collision detection.

One possible way to reduce the number of checks is to use a `quadtree` as the data stucture to store the ball objects. The basic idea of this algorithm is to use a `quadtree` to divide a 2D region into more manageable parts and skip the detection among the pairs that are in two distant parts.

Each node in the `quadtree` represent a certain area in the 2D region and this area can be divided into four sub-areas that represented by four sub-nodes of this node. Each object that we need to coduct the

detection will then be put into one of these sub-nodes according to where it lies in the 2D region. Then, we can skip the detection among the pairs of objects that are in two distant parts, for intance, the objects in the top-left node cannot be colliding with the objects in the bottom-right node.



quadtree

> **Note:** You can find the above image and more information about **quadtree** here.

8. Improvement and Further Work

Collision detection

As mentioned above, the computational complexity in collision detection can can be improved effectively if the `quadtree` data structure is adopted

Color Generation

One algorithm to generate color randomly that we have tried is to mix a random color with white (255, 255, 255) to increase the lightness of the color:

```
color = Color4F(127+127*CCRANDOM_0_1(),
                127+127*CCRANDOM_0_1(),
                127+127*CCRANDOM_0_1(), 1.0);
```

However, some color that are too dark to be highlighted on the background are still observed. So In this game, 30 aesthetically-pleasing color that we chose are stored in the code and randomly picked to be the color of the characters. However, this is inconvenient for the further development especially when we want

to add more colors for more players and ball objects on a bigger map. It is easily to have duplicate color between the balls as well.

Therefore, a algorithm to generate aesthetically-pleasing color randomly is on the list of the further work. One possible solution is to use a golden ratio and one examples can be found at here.

Game difficulty levels

In each game mode, levels are set from easiest to hardest from Level 1 to Level n. In the initial state, only Level 1 is unlocked and only when the last level is passed will the following level be unlocked. Otherwise the player cannot enter the next level. Each level is set with different prescribed playing time (for time limited mode), target weight and number and initial weights of AI asteroids and Demon asteroid. All these factors should be taken into consideration to win the game, flexibly using skills -- speeding up or shooting to escape attacks from enemies and eat more asteroids.

User interface design

There is still room for improvement of graphic interface and audio effect. For example, the hero asteroid can be shown in a unique bitmap or the image of it could be uploaded by the player.

Multi-player mode

Current we only achieve the single-player mode, so multi-player mode could add more challenges to this game, in which the players can be partners or competitors, equal to the process of game theory.