MỤC LỤC

DANH MỤC HÌNH ẢNH

MỞ ĐẦU

1. Lý do chọn đề tài

Trong xu hướng phát triển của ngành công nghệ thông tin, làm việc và giải trí trên PC đang dần được thay thế, chuyển đổi lên các thiết bị di động bởi tính tất yếu của ứng dụng di động. Tuy nhiên thời điểm hiện tại, các lĩnh vực để phát triển ứng dụng trên di động chưa nhiều. Trong khi nhu cầu sử dụng thiết bị di động để giải trí, chơi Game ngày càng tăng.

Các nhà sản xuất thiết bị di động lớn hiện nay chọn Android làm hệ điều hành chủ đạo, số lượng người dùng và thiết bị ngày càng tăng dẫn đến nhu cầu lớn về phần mềm. Android có tính mở cao, được sự hỗ trợ từ cộng đồng.

Lập trình Game Android là hướng đi mới, phải vận dụng nhiều kiến thức trong CNTT. Từ đó thúc đẩy khả năng tự nghiên cứu và vận dụng những kiến thức đã học áp dụng vào thực tế.

Trên hệ điều hành Android đã có một số Game tương tự Game FLAT WORLD, tuy nhiên cách chơi còn đơn giản, chưa giới hạn thời gian, bản đồ không được tạo ngẫu nhiên nên gây nhàm chán khi chơi lại. Game FLAT WORLD được xây dựng khắc phục được các thiếu sót trên thêm vào một số chức năng mới.

2. Mục tiêu và nhiệm vụ nghiên cứu.

- Mục tiêu nghiên cứu: Xây dựng Game "FLAT WORLD" trên hệ điều hành Android.
- Nhiệm vụ nghiên cứu:
    - Thu thập, đọc tài liệu, nghiên cứu tìm hiểu cách làm việc và lập trình trên nền tảng Android.
    - Nghiên cứu sử dụng ngôn ngữ Java và các thư viện trong Android để lập trình Game.
    - Nghiên cứu các thuật toán sử dụng trong Game.
    - Phân tích thiết kế Game.

- Viết đề cương, báo cáo, cài đặt chương trình.

3. Đối tượng và phạm vi nghiên cứu.

- Đối tượng: Game "FLAT WORLD" trên hệ điều hành Android
- Phạm vi: Một số API thuộc lớp Canvas nhằm đồ họa 2D trong Android, cảm biến gia tốc, xử lý âm thanh. Cách xử lý đồ họa Game, các thuật toán xử lý va chạm đơn giản trong không gian 2D.

4. Phương pháp nghiên cứu.

- Đọc tài liệu về lập trình Game trên Android.
- Tham gia thảo luận, hỏi đáp các diễn đàn chuyên môn, tham khảo code ví dụ, hướng dẫn trên mạng Internet về thuật toán, cách xử lý các vấn đề gặp phải. Tham gia thảo luận cùng sinh viên có chung đề tài.
- Thực hiện lập trình game, rút ra kinh nghiệm từ thực tế.

5. Dự kiến kết quả.

- Chương trình demo chạy trên thiết bị thật.

6. Ý nghĩa khoa học và thực tiễn.

- Sản phẩm là một trò chơi cho điện thoại chạy Android. Có thể upload lên chợ ứng dụng Android Market.

7. Nội dung đồ án

CHƯƠNG I: Cơ sở lý thuyết.

CHƯƠNG II: Phân tích, thiết kế hệ thống.

CHƯƠNG III: Chương trình demo.

Kết luận và hướng phát triển đề tài.

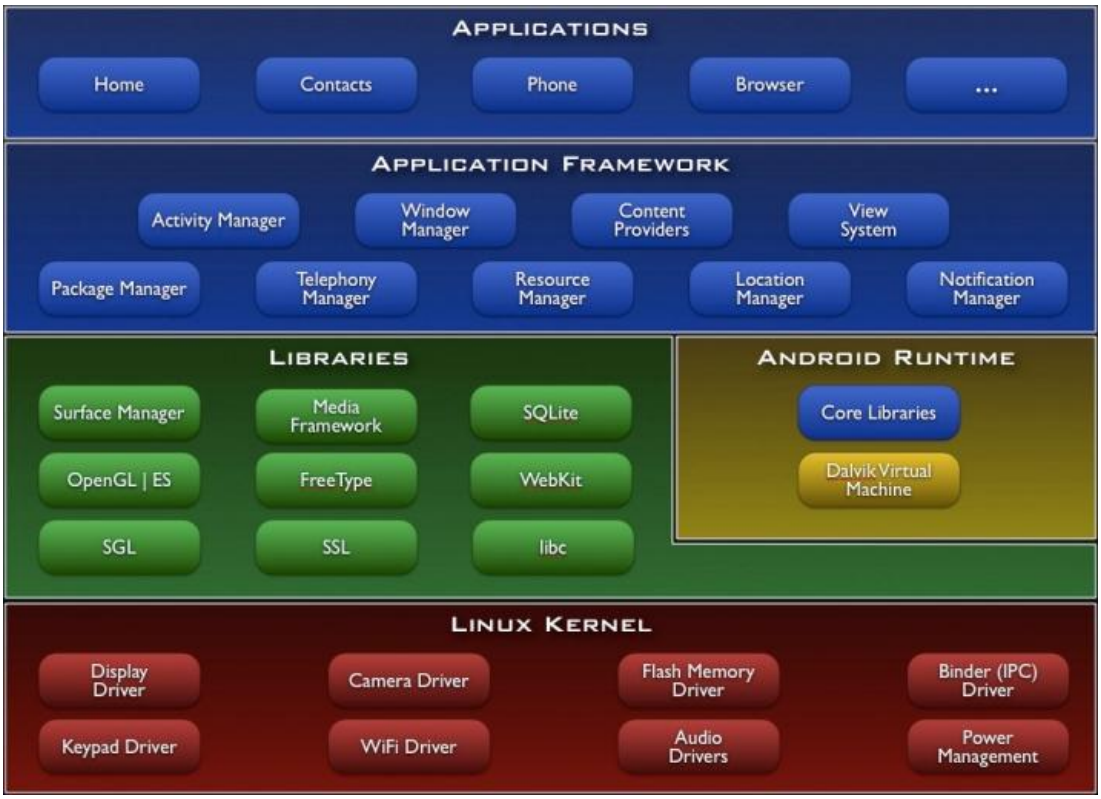
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Tổng quan Hệ điều hành Android và Game

1.1.1 Khái niệm Hệ điều hành Anroid

Android là hệ điều hành điện thoại di động mở nguồn mở miễn phí do công ty Google phát triển dựa trên nền tảng của Linux. Bất kỳ một hãng sản xuất phần cứng nào cũng đều có thể tự do sử dụng hệ điều hành Android cho thiết bị của mình, miễn là các thiết bị ấy đáp ứng được các tiêu chuẩn cơ bản do Google đặt ra (có cảm ứng chạm, GPS, 3G,...) Ra đời tháng 11/2007, hệ điều hành Android đã trải qua nhiều lần cập nhật, với phiên bản gần đây nhất là Ice Cream Sandwich 4.0. Android là nền tảng cho thiết bị di động bao gồm một hệ điều hành, midware và một số ứng dụng chủ đạo. Bộ công cụ Android SDK cung cấp các công cụ và bộ thư viên các hàm API cần thiết để phát triển ứng dụng cho nền tảng Android sử dụng ngôn ngữ lập trình java.

1.1.1.1 Kiến trúc Android



Hình 1 : Mô hình kiến trúc nền tảng Android

Theo tài liệu được cung cấp bởi Goolge, kiến trúc Android có 5 thành phần được được phần lớp từ cao xuống thấp. Lần lượt như sau.

❖ **Applications**

Hệ điều hành Android tích hợp sẵn một số ứng dụng cơ bản như email client, SMS, lịch điện tử, bản đồ, trình duyệt web, sổ liên lạc và một số ứng dụng khác. Ngoài ra tầng này cũng chính là tầng chứa các ứng dụng được phát triển bằng ngôn ngữ Java.

❖ **Application Framework**

Tầng này của hệ điều hành Android cung cấp một nền tảng phát triển ứng dụng mở qua đó cho phép nhà phát triển ứng dụng có khả năng tạo ra các ứng dụng vô cùng sáng tạo và phong phú. Các nhà phát triển ứng dụng được tự do sử dụng các tính năng cao cấp của thiết bị phần cứng như: thông tin định vị địa lý, khả năng chạy dịch vụ dưới nền, thiết lập đồng hồ báo thức, thêm notification vào status bar của màn hình thiết bị.

❖ **Libraries**

Hệ điều hành Android bao gồm một tập các bộ thư viện C/C++ được sử dụng bởi nhiều thành phần của Android system. Những tính năng này được cung cấp cho các lập trình viên thông qua bộ framework của Android. Dưới đây là một số thư viện cốt lõi:

❖ **Android Runtime**

Hệ điều hành Android tích hợp sẵn một tập hợp các thư viện cốt lõi cung cấp hầu hết các chức năng có sẵn trong các thư viện lõi của ngôn ngữ lập trình Java. Mọi ứng dụng của Android chạy trên một tiến trình của riêng nó cùng với một thể hiện của máy ảo Dalvik. Máy ảo Dalvik thực tế là một biến thể của máy ảo Java được sửa đổi, bổ sung các công nghệ đặc trưng của thiết bị di động. Nó được xây dựng với mục đích làm cho các thiết bị di động có thể chạy nhiều máy ảo một cách hiệu quả. Trước khi thực thi, bất kì ứng dụng Android nào cũng được convert thành file thực thi với định dạng nén Dalvik Executable (.dex). Định dạng này được thiết kế để phù hợp với các thiết bị hạn chế về bộ nhớ cũng như tốc độ xử lý. Ngoài ra máy ảo Dalvik sử dụng bộ nhân Linux để cung cấp các tính năng như thread, low-level memory management.

❖ **Linux Kernel**

Hệ điều hành Android được xây dựng trên bộ nhân Linux 2.6 cho những dịch vụ hệ thống cốt lõi như: security, memory management, process management, network stack, driver model. Bộ nhân này làm nhiệm vụ như một lớp trung gian kết nối phần cứng thiết bị và phần ứng dụng.

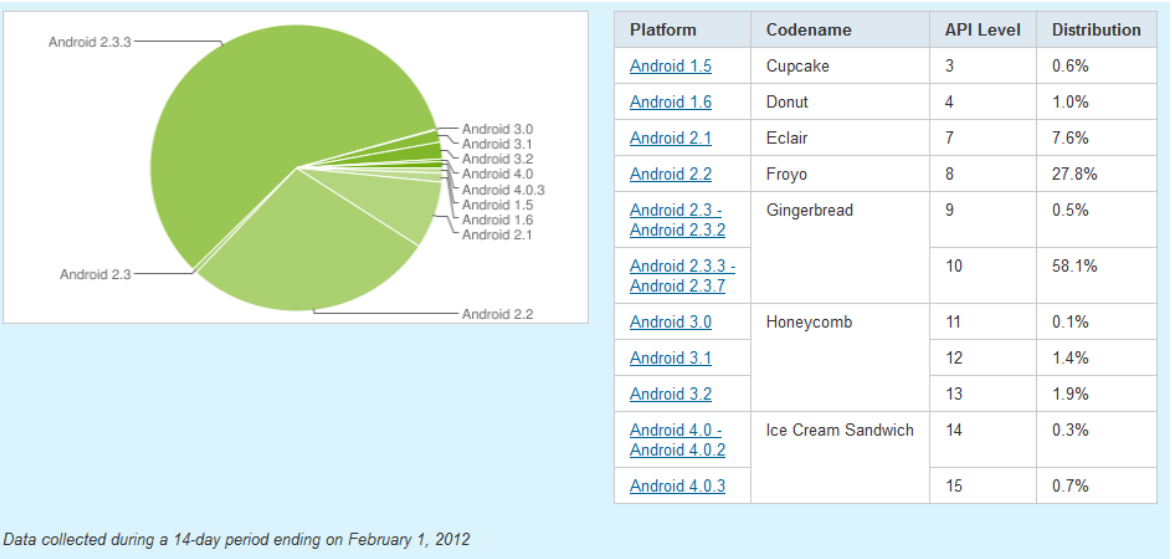
1.1.1.2 Phát triển ứng dụng trên Android

❖ **Ngôn ngữ lập trình**

Ngôn ngữ lập trình chính thức của Android là Java. Mặc dù các ứng dụng trên Android được phát triển dựa trên nền tảng Java, nhưng Android không hỗ trợ J2ME và J2SE, là hai ngôn ngữ lập trình phổ dụng cho các thiết bị di động.

Dựa trên máy ảo Java của Sun, Google đã tinh chỉnh và phát triển nên máy ảo Dalvik để biên dịch mã Java với tốc độ biên dịch nhanh hơn và nhẹ hơn. Đến phiên bản Froyo 2.2, Android đã hỗ trợ Just-in-time Compiler (JIT) làm tăng tốc độ biên dịch Java lên gấp 2-5 lần so với các phiên bản trước.

❖ **Các phiên bản Android**



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.6%
Android 1.6	Donut	4	1.0%
Android 2.1	Eclair	7	7.6%
Android 2.2	Froyo	8	27.8%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7		10	58.1%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.4%
Android 3.2		13	1.9%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.3%
Android 4.0.3		15	0.7%

Data collected during a 14-day period ending on February 1, 2012

Hình 2: Các phiên bản Android

Có nhiều phiên bản Android, phổ biến nhất là 2.3.3. Khi làm Game chú ý tới phiên bản hỗ trợ để chạy được trên nhiều phiên bản nhất.

❖ **Lập trình trên Android**

Lập trình trên Android cần các công cụ sau: Android SDK: bao gồm các công cụ riêng lẻ như: debugger, các thư viện, trình giả lập điện thoại Android, các tài liệu hỗ trợ và code mẫu. Hiện Android cung cấp bộ công cụ này trên nhiều nền tảng hệ điều hành khác nhau (Windows, Linux, Mac,...) yêu cầu cài sẵn Java Development Kit, Apache Ant và Python2.2 trở lên.

IDE (Môi trường phát triển tích hợp): Eclipse phiên bản 3.2 trở lên với plugin ADT (Android Development Tools), Netbeans. Tuy nhiên, người lập trình có thể sử dụng bất kỳ 1 IDE hay trình soạn thảo văn bản nào để viết code Java và XML rồi biên dịch nên ứng dụng hoàn chỉnh bằng cách sử dụng dòng lệnh (command lines). Eclipse hỗ trợ tốt hơn Netbeans nên được sử dụng làm công cụ phát triển chính thức.

Một số công cụ hỗ trợ lập trình Android tiêu biểu:

SQLite Manager: Là một addon của Firefox giúp quản lí SQLite của Android.

DroidDraw: Giúp thiết kế file XML giao diện ứng dụng.

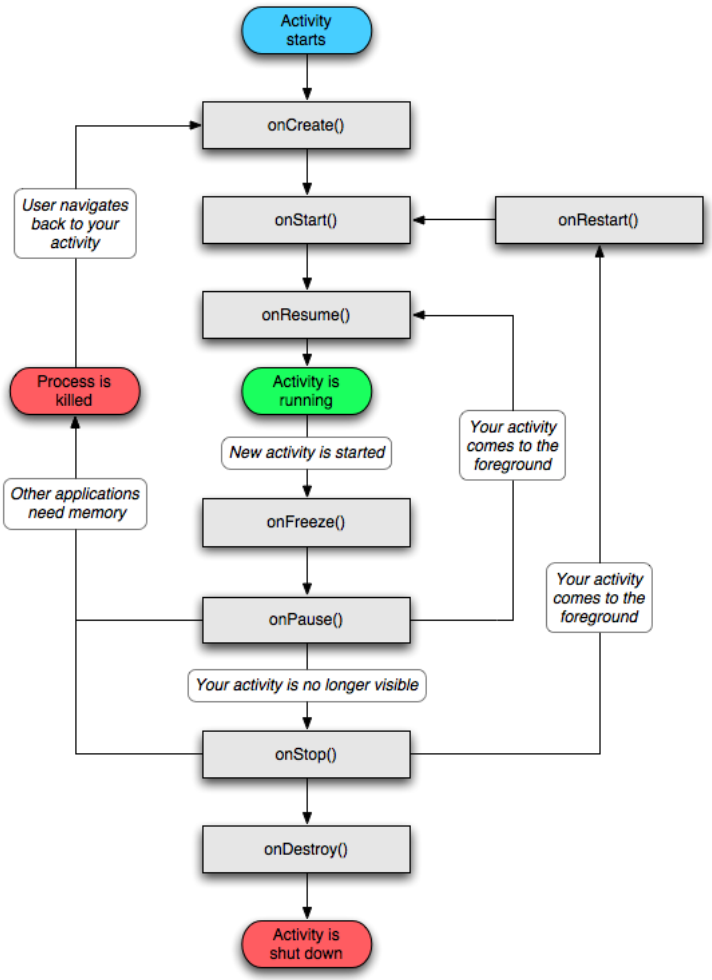
Balsamiq Mockups và AdobeFireworks: Giúp nhanh chóng phác thảo ý tưởng và giao diện sơ bộ của ứng dụng.

StarUML: Vẽ các lược đồ UML hỗ trợ phân tích thiết kế.

❖ **Chu kỳ sống của một ứng dụng Android:**

Trong mỗi ứng dụng Android có chứa nhiều thành phần và mỗi thành phần đều có một chu trình sống riêng. Và ứng dụng chỉ được gọi là kết thúc khi tất cả các thành phần trong ứng dụng kết thúc. Activity là một thành phần cho phép người dùng giao tiếp với ứng dụng. Tuy nhiên khi tất cả các Activity kết thúc và người dùng không còn giao tiếp được với ứng dụng nữa nhưng không có nghĩa là ứng dụng đã kết thúc. Bởi vì ngoài Activity là thành phần có khả năng tương tác với người dùng thì còn có các thành phần không có khả năng tương tác với người dùng như là Service, Broadcast receiver. Có nghĩa là thành phần không tương tác với người dùng có thể chạy nền dưới sự giám sát của HDH cho đến khi người dùng tự tắt chúng.

Thời gian sống của Activity:



Hình 3: Lược đồ vòng đời của 1 activity

Dựa vào lược đồ trên, thấy được có 3 vòng lặp quan trọng sau:

Vòng đời toàn diện (Entire Lifetime): Diễn ra từ lần gọi onCreate(Bundle) đầu tiên và kéo dài tới lần gọi onDestroy() cuối cùng.

Vòng đời thấy được (Visible Lifetime): Diễn ra từ khi gọi *onStart()* và kéo dài tới khi gọi *onStop()*. Ở vòng đời này, activity được hiển thị trên màn hình nhưng có thể không tương tác với người dùng ở trên nền. Các phương thức *onStart()* và *onStop()* có thể được gọi nhiều lần.

Vòng đời trên nền (Foreground Lifetime): Diễn ra từ khi gọi *onResume()* và kéo dài tới khi gọi *onPause()*. Ở vòng đời này, activity nằm trên mọi activity khác và tương tác được với người dùng. 1 activity có thể liên tục thay đổi giữa 2 trạng thái paused và resumed, chẳng hạn khi thiết bị sleep hay 1 intent mới được đưa tới.

Các hàm thực thi của vòng đời Activity:

OnCreate(): hàm này được gọi khi lớp Activity được gọi, dùng để thiết lập giao diện ứng dụng và thực thi các thao tác cơ bản.

Onstart(): hàm này được gọi khi lớp ứng dụng xuất hiện trên màn hình.

OnResume(): hàm được gọi ngay sau Onstart hoặc khi người dùng focus ứng dụng, hàm này sẽ đưa ứng dụng lên top màn hình.

OnPause(): hàm được gọi khi hệ thống đang focus đến một activity trước đó.

OnStop: hàm được gọi khi một Activity khác được khởi động và focus.

OnRestart(): được gọi khi ứng dụng chuyển sang *onStop()*, nhưng muốn khởi động lại bằng *onStart()*.

1.1.1.3 Giới thiệu cảm biến gia tốc

Cảm biến gia tốc nằm trong nhóm cảm biến chuyển động của điện thoại là một thiết bị phần cứng được hệ điều hành hỗ trợ một bộ API tiếp xúc với phần mềm, gọi là các API của cảm biến gia tốc, thông qua bộ giao diện này, phần mềm có thể xác định được điện thoại đang nghiêng theo trục nào (bao gồm cả gia tốc).

Hình 4: Các trục cảm biến gia tốc trên thiết bị di động

Giá trị nhận được từ cảm biến gian tốc cân bằng tại 0 khi điện thoại được đặt cân đối, để ngửa trên mặt bàn phẳng.

- Giá trị gia tốc x biến thiên từ trái qua phải thiết bị với giá trị -9.8 → 9.8
- Giá trị gia tốc y biến thiên từ sau về trước thiết bị với giá trị -9.8 → 9.8
- Giá trị gia tốc z biến thiên từ úp sang ngửa thiết bị với giá trị từ -9.8 → 9.8

Các giá trị được cập nhật liên tục với thời gian thực, như vậy ta có thể xác định hướng chuyển động của điện thoại dựa vào giá trị gia tốc x, y, z biến thiên để áp dụng vào Game.

1.1.2 Tổng quan về Game

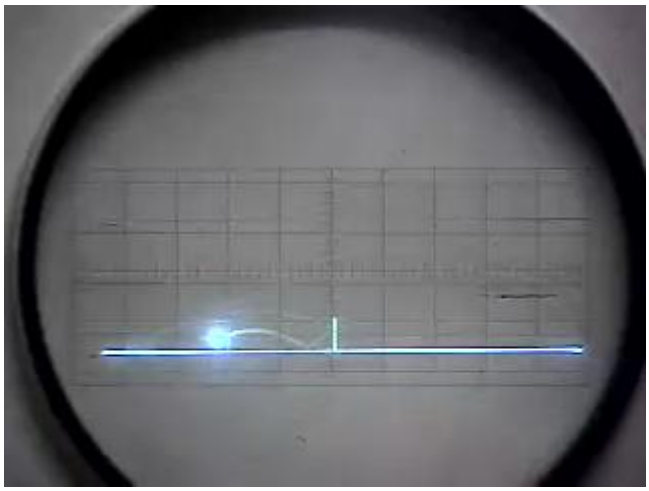
1.1.2.1 Khái niệm Game - video game:

Video game (gọi tắt là game) là một dạng trò chơi điện tử liên quan đến tính tương tác với một giao diện người sử dụng để tạo ra một phản hồi hình ảnh trên một thiết bị hiển thị (*video*).

❖ **Lịch sử Game:**

Vào những năm 1949–1950, Charley Adama chạy một chương trình có tên "Bouncing Ball" cho chiếc máy tính Whirlwind của MIT. Dù chương trình vẫn chưa thực sự tương tác được, nhưng nó đã là tiền thân của video game sắp ra đời. Phần lớn các game máy tính thời kỳ đầu đều chủ yếu sử dụng nội bộ trong các máy tính lớn ở các trường đại học và các phòng thí nghiệm và được phát triển bởi các cá nhân như một sở thích. Một số Game phổ biến ở thời kì này: Mouse in the Maze(Chuột trong mê cung), HAX.

Những năm 70: Vào năm 1971, Nolan Bushnell và Ted Dabney tạo ra một phiên bản trò chơi *Spacewar!* trên hệ máy arcade và gọi là *Computer Space.Là game thương mại chào bán lần đầu tiên.Một số game phổ biến cho thời kì này:Pong của Atari.* Thành công thương mại của trò Pong dẫn đến rất nhiều các công ty phát triển các bản sao của trò này trên các hệ máy của mình- sự sinh sôi của ngành công nghiệp game.



Hình 5: Game Tenis for Two - 1958

Những năm 80: Có sự đổi mới về thể loại của Game. Ra đời nhiều thể loại Game: Phiêu lưu hành động (*The Legend of Zelda* (1986)), game mê cung (PacMan), Game nhập vai (*Dragon Warrior* (1986)), game đua xe (Turbo(1981))…

Game trên máy tính: Năm 1984, thị trường game máy tính tiếp quản luôn thị trường game trên console do cơn khủng hoảng vào năm đó; máy tính cung cấp khả năng chơi game gần bằng với console và từ khi có kiểu thiết kế đơn giản của máy tính gia dụng cho phép phần mềm game có thể hoàn toàn điều khiển phần cứng sau khi bật máy (gần như game trở thành hệ điều hành), điều này khiến cho việc khởi động game trên máy tính dễ dàng như trên console.

Game online thời kì đầu: Hệ thống bảng thông tin (bulletin board systems) quay số (dial-up) đã khá phổ biến vào thập niên 80, và đôi khi được sử dụng để chơi game online.Một số Game online đầu tiên: *Snipes* (chim dẽ giun hay chim mỏ nhát), một game chơi mạng dạng chữ ra đời năm 1983 để kiểm tra một loại máy tính cá nhân mới của IBM và tính tương thích với hệ thống mạng của nó.

Cuộc khủng hoảng của video Game năm 1983:

Vào cuối năm 1983, ngành công nghiệp game trải qua sự tổn thất còn lớn hơn cả cuộc khủng hoảng năm 1977. Đây là sự sụp đổ của cả ngành công nghiệp (không giống như năm 1977 là chỉ sụp đổ cho hệ arcade và một phần console), cũng như là sự phá sản của một vài hãng chuyên sản xuất máy tính gia dụng và hệ console cho thị

trường Bắc Mỹ vào cuối năm 1983 và 1984. Nó mang đến cái gọi là sự kết thúc của các hệ máy console thế hệ hai.

Những năm 1990: Những năm 1990 đánh dấu một sự cách tân trong ngành công nghiệp game. Đó là thập kỷ của sự biến đổi từ đồ họa raster sang đồ họa 3 chiều và đã dẫn đến sự ra đời của một vài thể loại game bao gồm bắn súng góc nhìn người thứ nhất, chiến lược thời gian thực, và game trực tuyến. Game trên hệ máy cầm tay trở nên phổ biến rộng rãi, một phần nhờ vào việc phát hành Game Boy. Game trên arcade, mặc dù vẫn còn tương đối phổ biến vào đầu thập kỷ, bắt đầu suy giảm do sự phổ biến dần của các hệ console gia dụng.

Năm 1992, Dune II ra mắt. Game này không phải là game theo thể loại chiến lược thời gian thực đầu tiên .Nhưng Dune II đã đặt tiêu chuẩn về lối chơi cho các tựa game bom tấn về sau như *Warcraft: Orcs & Humans*, Command & Conquer, và StarCraft.

Game trên điện thoại di động: Điện thoại di động trở thành một thiết bị chơi game từ khi Nokia cài đặt *Snake* (rắn săn mồi) vào các dòng điện thoại của mình từ năm 1998. Game trên điện thoại di động thời kỳ đầu khá hạn chế ở một số điểm như: kích thước khiêm tốn của màn hình, tất cả đều là các game đơn sắc và sự giới hạn bộ nhớ cũng như sức mạnh xử lý, chưa tính đến việc tiêu thụ pin nhanh chóng khi chơi.

Các hệ máy console thế hệ thứ năm ra đời: Năm 1994, ba máy console mới được giới thiệu tai Nhật Bản: Sega Saturn, PlayStation, và PC-FX. *Resident Evil là game hay nhất nhất trên* PlayStation thời kì này.

Những năm 2000 cho đến nay: Hiện tượng người dùng tạo ra các bản chỉnh sửa (hay "mod") cho các game dần trở thành một xu hướng quanh thời điểm chuyển giao giữa thiên niên kỷ.

Game trên điện thoại di động: Game trên điện thoại di động bắt đầu được quan tâm khi Nokia giới thiệu điện thoại N-Gage là thiết bị chơi game cầm tay của mình vào năm 2003. Điện thoại di động chơi game có doanh thu hơn 1 tỷ USD vào năm 2003, và vượt qua 5 tỷ USD trong năm 2007, chiếm 1/4 doanh thu của tất cả video game trên thị trường. Các điện thoại đời mới hơn được bán ra thị trường như

các smartphone N-Series của Nokia trong năm 2005 và iPhone của Apple trong năm 2007 là những ví dụ mạnh mẽ cho sự hấp dẫn của game trên điện thoại di động. Năm 2008, Nokia không định phục hồi lại thương hiệu N-Gage nhưng lại cho ra một thư viện các game cho dòng điện thoại cao cấp này. Cũng trong năm 2008 trên *AppStore* của Apple, hơn một nửa trên tổng số tất cả các ứng dụng được bán ra là các game cho iPhone.

Thế hệ console thứ sáu:Sự ra đời Playstation 2 của Sony và gần cuối năm 2001, tập đoàn Microsoft, được biết đến nhiều nhất với hệ điều hành Windows và chuyên sản xuất các phần mềm, bước vào thị trường console với máy Xbox. Ngay sau đó vào tháng 12 năm 2001, studio phát triển game Bungie Studio cho ra đời *Halo: Combat Evolved* và ngay lập tức trở thành bước ngoặt quan trọng trong sự thành công của Xbox, và dòng game *Halo* sau đó trở thành một trong những game bắn súng trên console thành công nhất mọi thời đại.

Game trực tuyến tăng trưởng một các mạnh mẽ: Khi việc kết nối Internet băng thông rộng với giá cả phải chăng trở nên phổ biến, nhiều nhà phát hành quay sang game online như một cách để đổi mới.

Sự lên ngôi của các Game casual(game đơn giản): Khởi đầu với máy tính cá nhân, một xu hướng mới với game đơn giản, các trò chơi với sự phức tạp vừa phải được thiết kế để có thể chơi các phần một cách nhanh chóng hoặc ngẫu hứng, bắt đầu nhận được sự chú ý của ngành công nghiệp. Rất nhiều các game này chỉ là giải câu đố, xếp hình, như *Bejeweled* (xếp kim cương) của PopCap Games và *Diner Dash* của PlayFirst, trong khi các game khác với lối chơi thoải mái hơn và có kết thúc mở. Tựa game lớn nhất trong thể loại này là *The Sims* của hãng Maxis, mà trở thành game máy tính bán chạy nhất mọi thời đại, vượt qua cả tựa game phiêu lưu kinh điển *Myst*, cùng với rất nhiều các game khác, mà được gắn liền vào các trang mạng xã hội như Myspace hay Facebook.

Thế hệ console thứ bảy:Sony cho ra mắt Playstation3. Tập đoàn Apple bước vào lĩnh vực phần cứng chơi game di động với sự ra đời App Store cho iPhone và iPod Touch trong mùa hè năm 2008.

❖ **Các thể loại game:**

Game cốt lõi: Nhìn chung, thảo luận về game trên báo chí và chính trị chỉ xoay quanh các tựa game của game cốt lõi; bao gồm video game phát triển để chơi trên máy tính cá nhân, hệ console và hệ cầm tay. Game cốt lõi thường được xác định bởi cường độ của chúng, chiều sâu của lối chơi hay quy mô của sản phẩm liên quan đến quá trình tạo game và có thể bao gồm các game trên một phổ rộng các thể loại. Ví dụ dòng game Bit.Trip trên hệ thống mạng WiiWare, dòng Fallout cho PC và hệ console hay LittleBigPlanet cho hệ PS3, tất cả đều nằm trong phân loại game cốt lõi. Game cốt lõi đôi khi được xem xét ở góc độ lối chơi khá phức tạp và thường không hấp dẫn các người chơi thông thường (người chơi game để giải trí).

Game casual: Casual games (game đơn giản) có cái tên này do tính dễ dàng tiếp cận của chúng, lối chơi dễ hiểu và nắm bắt quy luật chơi nhanh chóng. Thêm vào đó , các game casual thường hỗ trợ khả năng tham gia vào và thoát ra nhanh theo yêu cầu. Game casual là một thể loại tồn tại rất lâu trước khi thuật ngữ xuất hiện và bao gồm cả các game như Solitaire hay Minesweeper mà thường thấy cài đặt sẵn trong các hệ điều hành Windows của Microsoft. Thí dụ cho thể loại này như là tìm vật ẩn, câu đố, quản lý thời gian, xếp hình-tetris hoặc rất nhiều các thể loại game chiến thuật chạy trên nền flash khác. Game casual thường được bán qua các dịch vụ bán lẻ trên mạng như PopCap, Zylom và GameHouse hoặc cung cấp miễn phí qua các trang web như Newgrounds hay AddictingGames.

Game nghiêm túc: Game nghiêm túc là những game được thiết kế chủ yếu để truyền đạt thông tin hoặc để học hỏi kinh nghiệm nào đó cho người chơi. Một số game nghiêm túc thậm chí không đủ điều kiện là một video game theo ý nghĩa truyền thống của thuật ngữ. Ngoài ra, phần mềm giáo dục không hẳn đã thuộc thể loại game này (ví dụ phần mềm hướng dẫn đánh máy, học ngoại ngữ, v.v...) và sự khác biệt chính sẽ dường như dựa trên mục đích chính của tiêu đề cũng như độ tuổi mà phần mềm hướng tới. Như với các thể loại khác, sự mô tả này giống một hướng dẫn hơn là một quy định. Game nghiêm túc là các game thường làm ra với các mục đích ngoài giải trí đơn thuần. Và như game nòng cốt và game casual, game nghiêm túc có thể bao gồm hoạt động ở bất kỳ thể loại nào , mặc dù một số thể loại như game luyện tập, game giáo dục, hay game tuyên truyền có thể có mặt nhiều hơn trong nhóm game nghiêm túc do

mục đích chuyên môn của chúng. Có những game thường được thiết kế để chơi bởi một chuyên gia như một phần của một công việc cụ thể hoặc để tăng cường kỹ năng. Chúng cũng có thể được tạo ra để truyền đạt nhận thức chính trị xã hội trên một phương diện cụ thể. Một trong những game nghiêm túc có thương hiệu lâu đời nhất có lẽ là Microsoft Flight Simulator (game mô phỏng lái máy bay) lần đầu phát hành vào năm 1982 với tên này. Quân đội Mỹ sử dụng game mô phỏng thục tế ảo này để làm bài tập huấn luyện, cùng với đó là sự phát triển một số lớn các phần hỗ trợ bay khác như lái máy bay cảnh sát, máy bay chiến đấu, máy bay y tế.

- **Ngôn ngữ lập trình Game:**

Tùy vào từng loại game cụ thể mà sử dụng các ngôn ngữ khác nhau. Đối với những game lớn giành cho PC và các hệ máy console thì C++ và DirectX hoặc OpenGL là lựa chọn tốt nhất vì đòi hỏi phải can thiệp sâu bên trong hệ thống,Java là ngôn ngữ lập trình game phổ biến cho các thiết bị di động và không phụ thuộc vào nền tảng.Ngoài ra thì còn có C# và Python…

Một số công cụ xây dưng game chuyên nghiệp:

Nếu như trước kia việc lập trình game phải làm thủ công các công đoạn từ đầu tới cuối từ việc xây dựng đồ họa đến âm thanh ,…thì hiên nay với sự ra đời nhiều công cụ hỗ trợ giúp việc lập trình game nhanh hơn và tối ưu hơn gọi là game engine.

Một **game engine** (công cụ tạo game) là một phần mềm được viết để thiết kế và phát triển video game. Có rất nhiều loại game engine dùng để thiết kế game cho các hệ máy như hệ consoles hay máy tính cá nhân(PC). Chức năng cốt lõi của game engine phần lớn nằm trong công cụ dựng hình (kết xuất đồ họa) cho các hình ảnh 2 chiều (2D) hay 3 chiều(3D), công cụ vật lý (hay công cụ tính toán và phát hiện va chạm), âm thanh, mã nguồn, hình ảnh động (animation), trí tuệ nhân tạo, phân luồng, tạo dòng dữ liệu xử lý , quản lý bộ nhớ, dựng ảnh đồ thị, và kết nối mạng. Quá trình phát triển game tiết kiệm được rất nhiều thời gian và kinh phí vào việc tái sử dụng và tái thích ứng một engine để tạo nhiều game khác nhau.

Với công nghệ tạo game engine càng phát triển và trở nên thân thiện hơn với người sử dụng ,ứng dụng của nó càng được mở rộng, và giờ đây được sử dụng để tạo các game mang mục đích khác với giải trí đơn thuần như: mô phỏng, huấn luyện ảo, y

tế ảo, và mô phỏng các ứng dụng quân sự. Để tạo điều kiện tiếp cận này, các nền tảng phần cứng mới đang là mục tiêu của game engine, kể cả điện thoại di động (ví dụ: iPhone, điện thoại Android) và trình duyệt web (như Shockwave, Flash, WebVision của Trinigy, Silverlight, Unity Web Player, O3D và thuần dhtml). Thêm vào đó, nhiều game engine đang được tạo ra bằng các ngôn ngữ lập trình cấp cao như Java và C# hay .NET (ví dụ TorqueX, và Visual3D.NET) hay Python (Panda3D). Vì hầu hết các game 3D hiện nay đều có giới hạn cho GPU(giới hạn bởi sức mạnh của card đồ họa), khả năng gây chậm máy của các ngôn ngữ lập trình cấp cao trở nên không đáng kể, trong khi việc tăng năng suất được cung cấp bởi các ngôn ngữ này lại có lợi cho các nhà phát triển game engine.

1.1.2.2 Lý thuyết về lập trình Game

Khác với lập trình giao diện trên Android sử dụng các API để tạo các Button, Label, Text View… Lập trình Game đòi hòi người lập trình phải vẽ nên tất cả giao diện mà không sử dụng các API được hệ điều hành hỗ trợ.

Lập trình Game thường tuân theo 1 Frame work, tạo ra 1 bộ Interface để Implement vào Game cụ thể và tùy biến theo yêu cầu, dưới Hệ điều hành nào cũng cần các API sau để tạo nên Game:

- **Window management**: Quản lý tạo, đóng cửa sổ Game, theo dõi tiến trình hay pause/resume game trên Anroid.
- **Input**: Quản lý các thành phần liên quan đến nhập dữ liệu (Nhấn phím, chạm vào màn hình, cảm biến…)
- **File I/O**: Quản lý nhập xuất file
- **Audio**: tải tập tin âm thanh, quản lý tiến trình sử dụng âm thanh (Play/Pause).
- **Graphics**: Thành phần quan trọng nhất. Tải các đối tượng đồ họa và vẽ chúng lên màn hình.
- **Game Framework**: Đưa tất cả các thành phần trên vào 1 khung, xây dựng các lớp để dễ dàng sử dụng và làm việc với mức cao hơn, tập trung vào xây dựng Game hơn là thao tác cấp thấp.

1.1.2.3 Cấu trúc một ứng dụng Game

❖ **Vòng lặp Game**

Một vòng lặp Game sẽ thực hiện tất cả các thao tác, tiến trình trong Game cho đến khi kết thúc, như dựng hình, render, update frame, xử lý âm thanh, xử lý tương tác, va chạm… Gọi là vòng lặp Game (Main Loop).

```
While(!userQuit)
    Kiểm_tra_sự_kiện_nhập();
    Chạy_AI();
    Di_chuyển_đối_tượng();
    Xử_lý_va_chạm();
    Vẽ_đối_tượng_lên_màn_hình();
    Phát_âm_thanh();
end While
```

Trong vòng lặp chính sẽ tùy vào tương tác của người chơi để chuyển đến các trạng thái tương ứng với các màn hình riêng nội tại. Có các trạng thái phổ biến như:

1) start up
2) licenses
3) introductory movie
4) front end
    1. game options
    2. sound options
    3. video options
5) loading screen
6) main game
    1. introduction
    2. game play

- game modes

3. pause options

7) end game movie
8) credits
9) shut down

Game được dựng hình dựa vào thời gian qua mỗi chu kỳ của Main Loop, thời gian này có thể không giống nhau vì phụ thuộc vào thời gian xử lý tương tác trong mỗi vòng lặp. 1 đại lượng là deltaTime. Main loop sẽ dựa vào deltaTime để tính cố frame cần render. Ta đề cập đến khái niệm khung khình trên giây.

❖ **FPS (Frame per seccond) – Khung hình trên giây**

Số lần vẽ khung hình trên một giây.FPS chậm sẽ làm cho người chơi không có cảm giác thật, ảnh hưởng đến sự tương tác của người dùng với trò chơi.

Các yếu tố làm cho FPS chậm:

- Hình vẽ có kích thước lớn.
- Tính toán trong game nhiều.
- Lỗi của thiết bị.

Cách cải thiện FPS:

- Tối ưu hóa tính toán và các điều kiện thực hiện trong Game:
  - Sử dụng ít vòng lặp.
  - Dùng phép toán dịch chuyển bit thay cho phép nhân và phép chia.
  - Khai báo các biến và phương pháp ở dạng static.
- Dùng Backbuffer: Không vẽ từng Sprite ra màn hình mà vẽ các Sprite ra một ảnh sau đó vẽ ảnh đó ra màn hình.

❖ **Bộ nhớ**

Là kích thước của bộ nhớ có thể dùng để chạy chương trình Game của thiết bị.

Các yếu tố ảnh hưởng đến bộ nhớ:

- File ảnh dùng trong Game.

- Âm thanh dùng trong Game.
- Các file text, mảng chứa text trong Game

1.1.2.4 Công cụ lập trình Game

1. SDK (Debuger, Simulator, Device Manager…): SDK được cung cấp bởi Google, với các trình hỗ trợ Debuger, máy ảo mô phỏng, trình quản lý thiết bị… Cung cấp các thư viện cũng như cập nhật các công cụ của SDK với phiên bản mới nhất.
2. IDE (Compiler, Editor, Library…): IDE được sử dụng là Eclipse phiên bản 3.2 trở lên, là một trình biên dịch code Java có cài thêm ADT Plugin để tích hợp SDK vào IDE. Eclipse được cung cấp bởi Oracle, được phát triển và nâng cấp đều đặn, phù hợp với các phiên bản Android được ra liên tục. Ngoài ra Eclipse còn hỗ trợ tìm lỗi code, điền sẵn, định dạng code…
3. Các phần mềm về đồ họa để tạo hình ảnh cho game: Photoshop, 3D max , Blender
4. Các phần mềm tạo, xử lý âm thanh, video, hiệu ứng…
5. Các Engine, tùy chọn có thể có hoặc không. Đa số các Game đều sử dụng engine để giảm thời gian thực hiện dự án.

❖ **Cài đặt môi trường lập trình cho Android**

Cài đặt môi trường Java

SDK và IDE cho Android đều có các module chạy trên Java. Đầu tiên cần cài môi trường Java vào máy tính. Tải bản mới nhất của JRE tại: http://www.oracle.com/technetwork/java/javase/downloads/index.html . Hoặc tải JDK để cài đặt đầy đủ công cụ phát triển Java, sau khi cài JDK sẽ tự cài JRE.

Cài đặt Eclipse

Hiện có nhiều IDE có khả năng hỗ trợ lập trình Android. Eclipse hỗ trợ đầy đủ nhất nên được lựa chọn làm IDE chính phát triển ứng dụng Android. Để cài đặt, đầu tiên tải phiên bản **Eclipse Classic** hoặc **Eclipse IDE for Java and Report Developer**

tại trang chủ: http://www.eclipse.org/downloads/ Sau khi tải về, giải nén để chạy Eclipse.

Cài đặt Android SDK

Vào trang http://developer.android.com/sdk/index.html download Android SDK, tùy thuộc vào hệ điều hành mà chọn bàn SDK cho phù hợp với máy. Sau khi cài đặt, từ Eclipse chọn Windows → Android SDK and AVD Manager → Available Packages để chọn các gói API cần cập nhật cho SDK. Chọn Install Selected. Chờ tiến trình tự động tải và cài đặt.

Cài ADT cho Eclipse

Cách 1: Cài trực tiếp từ internet:

Chạyt Eclipse, **Help** > **Install New Software...**.

Click **Add**

Dòng Location copy đường link ADT (lấy tại địa chỉ: http://dl-ssl.google.com/android/eclipse/)

Click **OK, Next. Finish.** Sau khi cài xong restart Eclipse.

Cách 2: Thay vì cài trực tiếp cài từ internet thì down file ADT về máy (không giải nén). Tại bước add vào Location thay vì copy link vào thì nhấn chọn Archive chọn tới file ADT down về.

Lưu ý: tùy từng Eclipse, tùy phiên bản 32 bit hay 64 bit thì các đường dẫn có thể khác nhau.

Kết luận: Chương 1 trình bày những hiểu biết cần thiết về Game và lập trình Game trên Android cũng như hiểu biết tổng quan về Android. Sau khi thu thập được tài liệu và có những kiến thức cần thiết, các hiểu biết về cung cụ và cách sử dụng công cụ. Sinh viên có thể tiến hành Phân tích thiết kế một ý tưởng Game. Nội dung được trình bày trong Chương 2.

CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

2.1 Giới thiệu Game

Game "FLAT WORLD" là một game tương tác với người sử dụng thông qua cảm biến gia tốc của điện thoại để điều khiển một viên bi lăn trên mặt bàn, vượt chướng ngại vật đến đích với thời gian và chướng ngại vật tăng dần tương ứng với cấp độ mà người chơi đang chơi.

Mỗi màn chơi được tạo ngẫu nhiên các chướng ngại vật, màn sau nhiều hơn màn trước, thời gian cũng tăng lên tương ứng, nếu người chơi đến đích trước khi hết thời gian sẽ thắng và qua màn tiếp theo, nếu va chạm phải chướng ngại vật, trò chơi sẽ kết thúc.

Thắng mỗi màn, người chơi sẽ được tính điểm. Điểm được lưu vào một danh sách mà người chơi có thể xem lại, mục đích của tính điểm để người chơi có thể cố gắng chơi tốt hơn, dành nhiều điểm hơn lần trước (thời gian còn lại càng nhiều thì điểm càng cao).

Trước khi đi vào thiết kế Game, cần thiết kế một Framework thực hiện các công việc ở mức thấp, sử dụng các API xây dựng các phương thức để phần thiết kế Game chỉ tập trung và phát triển Game mà không quan tâm nhiều đến các công việc ở mức thấp hơn.

2.2 Thiết kế Framework

Các modules cần phải có để xây dựng một Game:

- `Window management` (quản lý cửa sổ, tiếng trình): Quản lý các tác vụ như khởi tạo (create), đóng (close), tạm dừng (paused), chạy lại (resume) ứng dụng. Input (đầu vào): Quản lý, theo dõi các thao tác của người sử dụng để nhận dữ liệu đầu vào.
- `File I/O` (đọc ghi file): Cho phép đọc ghi file trên đĩa.
- `Graphics` (Đồ họa): Xử lý đồ họa, tải các tài nguyên và vẽ lên màn hình.
- `Audio` (Âm thanh): Tải, xử lý tiếng trình chơi âm thanh trong Game.

- `Game Framework`: Kết hợp những điều trên với nhau, cung cấp một nền tảng dễ sử dụng hơn để viết Game.

2.2.1 Window management

Với các phần mềm không phải là Game, giao diện được xây dựng bằng các API hỗ trợ từ hệ điều hành như tạo ra các nút bấm, các nhãn. Trong Game phải tự xây dựng tất cả các giao diện. Yêu cầu thêm nhiệm vụ của trình quản lý cửa sổ. Trình quản lý cửa sổ sẽ nhận tương tác từ người chơi, xác định tương tác đó là gì, từ đó biến đổi giao diện Game theo yêu cầu. Theo dõi các sự kiện, thay đổi kích thước màn hình. 3 phương thức liên quan của một ứng dụng Android mà Game cần trình quản lý cửa sổ theo dõi là:

onCreate(): Gọi 1 lần duy nhất khi khởi động ứng dụng

onPause(): Gọi khi ứng dụng bị tạm dừng

onResume(): Gọi khi ứng dụng được gọi trở lại từ dạng chạy nền.

2.2.2 Input

Trên hệ điều hành Android, có ba phương thức nhận tín hiệu đầu vào chính: Chạm màn hình, Nhấn phím và cảm biến gia tốc. Ứng dụng sẽ nhận tương tác của người chơi qua các sự kiện, có hai hình thức nhận sự kiện:

- Nhận tín hiệu ở thời điểm hiện tại: Hình thức áp dụng với cảm biến gia tốc, chỉ nhận giá trị tương tác tại thời điểm hiện tại.
- Nhận tín hiệu theo sự kiện: Các sự kiện được lưu lại vào một hàng đợi, chờ lấy ra xử lý tuần tự, áp dụng với sự kiện nhập liệu, bấm phím, chạm vào màn hình.

Hình thức chạm vào màn hình có 3 sự kiện:

- `Touch down:` Xảy ra khi người dùng nhấn xuống.
- `Touch up:` Xảy ra khi người dùng nhấc tay lên khỏi điểm chạm, luôn đi sau Touch down.
- `Touch Drag:` Xảy ra khi người dùng di chuyển điểm chạm trên màn hình.

Hình thức nhấn phím có 2 sự kiện

- `Key down`: Xảy ra khi phím được nhấn.
- `Key up:` Xảy ra khi ngừng nhấn phím, luôn đi sau Key down.

Hình thức Cảm biến gia tốc luôn phát tín hiệu theo sự thay đổi hướng của thiết bị.

Giao diện của Input được thiết kế với các lớp và phương thức:

```
public interface Input {
    public static class KeyEvent {}
    public static class TouchEvent {}
    public boolean isKeyPressed(int keyCode);
    public boolean isTouchDown(int pointer);
    public int getTouchX(int pointer);
    public int getTouchY(int pointer);
    public float getAccelX();
    public float getAccelY();
    public float getAccelZ();
    public List<KeyEvent> getKeyEvents();
    public List<TouchEvent> getTouchEvents();
}
```

class `KeyEvent:` Một thể hiện của một sự kiện nhấn phím.

class `TouchEvent:` Một thể hiện của một sự kiện chạm vào màn hình.

boolean `isKeyPressed(int keyCode):` theo dõi phím có keyCode có được nhấn không.

boolean `isTouchDown(int pointer):` theo dõi sự kiện chạm có đang diễn ra hay không.

int `getTouchX(int pointer):` lấy tọa độ x của điển chạm

int `getTouchY(int pointer):` lấy tọa độ y của điểm chạm

float `getAccelX():` nhận giá trị cảm biến gia tốc theo trục X

float `getAccelY():` nhận giá trị cảm biến theo trục Y

`float getAccelZ()`: nhận giá trị cảm biến theo trục Z

`List<KeyEvent> getKeyEvents()`: Danh sách sự kiện nhấn phím

`List<TouchEvent> getTouchEvents()`:Danh sách sự kiện chạm màn hình

2.2.3 File I/O

Thực hiện việc đọc và ghi file, đọc file nhị phân (file ảnh, âm thanh) từ thư viện tài nguyên Game (trong Android gọi là Asset). Việc đọc và ghi sử dụng thư viện `InputStream` và `OutputStream` của Java. File I/O có 3 phương thức:

Đọc dữ liệu từ Asset của Android.

```
public InputStream readAsset(String fileName) throws
IOException;
```

Đọc file.

```
public InputStream readFile(String fileName) throws
IOException;
```

Ghi file.

```
public OutputStream writeFile(String fileName) throws
IOException;
```

Việc đọc và ghi file thường có thể xảy ra lỗi nên sử dụng một ngoại lệ kiểm soát lỗi.

2.2.4 Audio

Phát âm thanh trong Game được chia làm 2 cách thực hiện:

- Phát một bản nhạc, thời gian chơi nhạc dài: nhạc nền.
- Phát một tiếng động, ngắn, nhanh: tiếng súng bắn, tiếng va chạm…

Với âm thanh dài, Game cần tải hết vào bộ nhớ, sau đó chơi đoạn âm thanh với một proces (tiến trình) riêng, tiến trình sẽ kết thúc khi Game được kết thúc hoặc tắt bởi mã lập trình sẵn. Quản lý tiến trình chơi nhạc bằng các phương thức: Play, pause, stop. Với âm thanh ngắn, dung lượng tương đối nhẹ, mỗi lần cần phát âm thanh có thể tải lại file từ đĩa. Frame work sẽ gồm các interface: Audio, Music, Sound để quản lý âm thanh.

2.2.4.1 Giao diện audio

```
public interface Audio {
     public Music newMusic(String filename);
     public Sound newSound(String filename);
}
```

Hai phương thức trên để tạo mới một tiến trình chơi nhạc và âm thanh.

2.2.4.2 Giao diện Music

Với giao diện này, Game cần các phương thức để phát, dừng, tạm dừng, thiết lập phát lặp lại, âm lượng, kiểm tra trạng thái tiến trình, hủy tiến trình. Giao diện được xây dựng với các phương thức như sau:

void play(): Phát âm thanh.
void stop(): Dừng phát âm thanh.
void pause(): Tạm dừng phát âm thanh.
void setLooping(**boolean** looping):Thiết lập âm thanh phát lặp lại.
void setVolume(**float** volume): Thiết lập âm lượng.
boolean isPlaying(): kiểm tra tiếng trình có đang chơi nhạc hay không.
boolean isStopped(): kiểm tra tiếng trình chơi nhạc đã dừng hay chưa.
boolean isLooping(): kiểm tra tiến trình có lặp lại không.
public void dispose(): Hủy tiến trình, giải phóng bộ nhớ.

2.2.4.3 Giao diện Sound

Với âm thanh đơn, cần 2 phương thức là phát âm thanh với âm lượng truyền vào và hủy âm thanh.

```
void play(float volume);
void dispose();
```

Khi cần tải âm thanh vào Game, phương thức newMusic và newSound sẽ lấy tên file được truyền vào và đọc file từ đĩa. File được lấy từ hệ thống thư viện tài nguyên Game trong Andoid (Asset).

2.2.5 Graphics

Để định nghĩa giao diện Graphics, cần nắm rõ một số hiểu biết liên quan:

- Raster - Lưới hiển thị
- Pixel - Điểm ảnh
- Framebuffers – bộ đệm khung hình
- Image Format – định dạng hình ảnh.

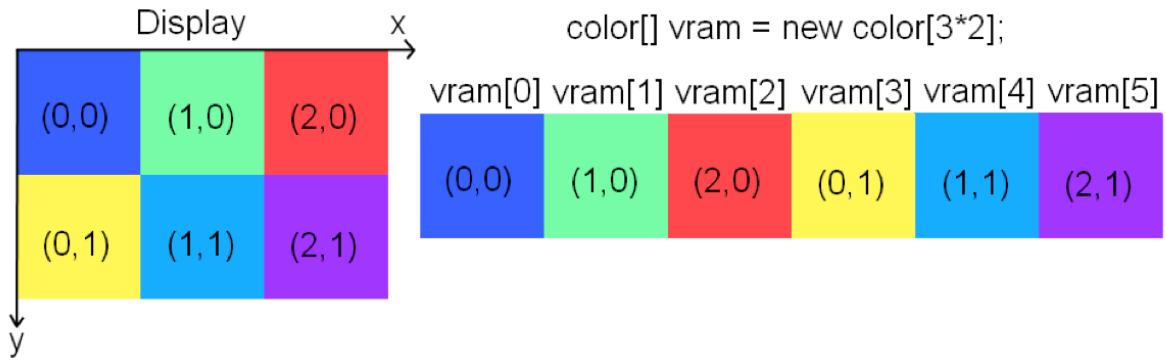
Raster: là một lưới hiển thị hình ảnh, một mảng 2 chiều có hữu hạn chiều dài và chiều rộng, mỗi ô lưới sẽ chứa một số lượng các điểm ảnh được hiển thị. Gọi là số Pixel trên hàng, và trên cột.

Pixel: là một điểm ảnh trên màn hình, đơn vị hiển thị nhỏ nhất. Có 2 thuộc tính là địa chỉ trên lưới hiển thị và màu sắc. Địa chỉ của Pixel được tính dựa trên tọa độ x, y của lưới, x là trục ngang, y là trục dọc tính từ góc trên, bên trái của màn hình (0,0).

Frame buffers: Tiến trình đồ họa truy cập một bộ nhớ đặc biệt gọi là bộ nhớ video (VRAM). VRAM là mảng 1 chiều được dùng để lưu trữ các điểm ảnh sẽ hiển thị trên màn hình, hiển thị một hình ảnh hoàn chỉnh. Trong thời gian một khung hình đang được vẽ lên màn hình, VRAM đã được chuẩn bị để tải khung hình tiếp theo vào.

Hệ tọa độ trong Game.

Thay vì tọa độ thông thường, được tính từ gốc (0,0) với trục y đi lên, trục x nằm ngang. Tọa độ trong Game sử dụng trục y đi xuống, do phải đọc từ đầu bộ nhớ VRAM (0,0).



Hình 6: Lưới hiển thị và VRAM

Góc trên cùng, bên trái của hình ảnh được đưa vào VRAM trước, nên khi lấy ra cũng phải lấy từ đầu mảng. Lấy điểm ảnh từ trái sang, từ trên xuống.

Image Format: Định dạng hình ảnh chất lượng tốt nhất là RGB888 tương ứng là R(màu đỏ), G(màu xanh lá), B (Màu xanh da trời) pha trộn cho màu của điểm ảnh đó, 888 là số bit cho mỗi điểm ảnh. Một điểm ảnh sẽ có 8+8+8 = 24 bit. Hình ảnh sẽ có kích thước lớn., vì vậy trong Game sử dụng định dạng ảnh nén là JPEG và PNG có định dạng thấp hơn. Khi tải hình ảnh vào bộ nhớ, Game sẽ sử dụng API cung cấp bởi hệ điều hành để thay đổi định dạng ảnh.

Ảnh có định dạng RGB sẽ không thể hiện được hình ảnh trong suốt (transparent), định dạng này được sử dụng để tải ảnh nền hoặc những hình ảnh phía dưới. Để tải ảnh có thể hiện được phần hình ảnh trong suốt phải sử dụng định dạng ARGB. Với A là giá trị alpha, thể hiện mức độ trong suốt của hình ảnh.

Hình 7: So sánh RGB và ARGB

Khi tải vào Game, hình ảnh cần nhẹ hơn nên sử dụng định dạng ARGB4444 cho mỗi điểm ảnh → 4+4+4+4 = 16 bit, và RGB565 → 5+6+5 = 16 bit (không có giá trị alpha) cho ảnh nền. Với những thông tin trên, ta thiết kế Interface Graphics thực hiện các công việc sau:

- Tải ảnh từ đĩa vào bộ nhớ và lưu trữ để sử dụng về sau.

- Tẩy frame buffer bằng 1 màu sắc bất kỳ, có thể tẩy hết các những gì còn sót lại từ frame trước.
- Thiết lập màu sắc và địa chỉ cho một điểm ảnh.
- Vẽ hình tròn, hình chữ nhật, đường thẳng.
- Vẽ hoàn toàn 1 hình ảnh hoặc 1 phần hình ảnh ra một vị trí xác định trên frame.
- Lấy kích thước của frame.

Graphics được thiết kế như sau:

```
public interface Graphics {
    public static enum PixmapFormat {
        ARGB8888, ARGB4444, RGB565}
    public Pixmap newPixmap(String fileName, PixmapFormat
    format);
    public void clear(int color);
    public void drawPixel(int x, int y, int color);
    public void drawLine(int x, int y, int x2, int y2,
    int color);
    public void drawRect(int x, int y, int width, int
    height, int color);
    public void drawPixmap(Pixmap pixmap, int x, int y,
    int srcX, int srcY, int srcWidth, int srcHeight);
    public void drawPixmap(Pixmap pixmap, int x, int y);
    public void drawPixmap(Pixmap pixmap, float x, float
    y);
    public int getWidth();
    public int getHeight();
}
```

Kiểu liệt kê các định dạng ảnh sẽ sử dụng.

```
static enum PixmapFormat {ARGB8888, ARGB4444, RGB565};
```

Phương thức khởi tạo 1 ảnh với tên file và định dạng.

```
Pixmap newPixmap(String fileName, PixmapFormat format);
```

Tẩy frame với màu truyền vào.

```
void clear(int color);
```

Vẽ một điểm ảnh.

```
void drawPixel(int x, int y, int color);
```

Vẽ một đường thẳng với tọa độ x1, x2, y1, y2 và màu.

```
void drawLine(int x, int y, int x2, int y2, int color);
```

Vẽ một hình chữ nhật với tọa độ x, y, rộng, dài và màu

```
void drawRect(int x, int y, int width, int height, int
color);
```

Vẽ một ảnh với tọa độ x, y. Tọa độ scrX, scrY bắt đầu trên ảnh nguồn, chiều rộng, chiều dài cần lấy.

```
void drawPixmap(Pixmap pixmap, int x, int y, int srcX, int
srcY, int srcWidth, int srcHeight);
```

Vẽ một ảnh với tọa độ x, y nguyên, thực.

```
void drawPixmap(Pixmap pixmap, int x, int y);
void drawPixmap(Pixmap pixmap, float x, float y);
```

Lấy chiều rộng frame.

```
int getWidth();
```

Lấy chiều dài frame.

```
int getHeight();
```

Hình ảnh cần một thể hiện để dễ dàng tương tác, ta thiết kế 1 Interface Pixmap cho phép lấy kích thước dài, rộng, định dạng, hủy ảnh khỏi bộ nhớ.

```
public interface Pixmap {
    public int getWidth();
    public int getHeight();
    public PixmapFormat getFormat();
    public void dispose();
}
```

2.2.6 Game Framework

Những nhiệm vụ của Game Framework để viết Game:

- Chia Game thành nhiều màn hình khác nhau với các tác vụ: Nhận dữ liệu, thao tác nhập từ user, áp dụng dữ liệu vào trạng thái của màn hình, và vẽ tất cả lên màn hình. Một số màn hình không cần dữ liệu từ user, nhưng sẽ tự động chuyển qua màn hình khác (ví dụ., màn hình Loading, Splash).
- Quản lý việc chuyển đổi màn hình, cung cấp phương pháp chuyển qua màn hình khác, hủy màn hình hiện tại và vẽ màn hình mới.
- Cho phép Game có quyền truy cập đến các modules khác (File I/O, Graphics, audio). Từ đó có thể tải các tài nguyên, lấy thông tin nhập vào, chơi âm thanh, vẽ lên màn hình…
- Game phải được tương tác như với thế giới thực, có nghĩa là mọi thứ sẽ được chuyển động và cập nhật liên tục. Màn hình Game luôn có khả năng cập nhật trạng thái và vẽ chính nó trong một vòng lặp gọi mà *MainLoop*, vòng lặp sẽ hủy khi thoát Game, mỗi vòng lặp được gọi là 1 khung. Số vòng lặp trên giây – FPS (Frame per second) mà chúng ta có thể tính toán gọi là *frame rate* (tốc độ khung hình).
- Theo dõi thời gian trôi qua từ lúc bắt đầu vẽ khung hình, được sử dụng để vẽ từng khung hình độc lập, gọi là đại lượng delta time.
- Theo dõi trạng thái của cửa sổ Game như một ứng dụng bình thường (khi nào bị paused hay resume) để xử lý sự kiện liên quan.
- Thiết lập cửa sổ và tạo giao diện người dùng sẽ nhận tương tác từ người sử dụng. Sau đó vẽ chúng lên và nhận sự kiện từ đó.

Main Loop (vòng lặp chính của Game) có dạng như sau:

```
while( !userQuit() ) {
        float deltaTime = currentTime() – lastFrameTime;
        lastFrameTime = currentTime();
        currentScreen.updateState(input, deltaTime);
```

```
            currentScreen.present(graphics, audio,
            deltaTime);
            }

        cleanupResources();
```

Game sẽ thoát khi nhận yêu cầu thoát của người chơi. Mọi vòng lặp Game đều tính một lượng thời gian delta time là thời gian trôi qua giữa các vòng lặp. Lấy thời gian hiện tại frame đang chạy trừ đi thời gian kết thúc frame trước ta được thời gian delta. Thuật toán tính delta time được trình bày trong phần thiết kế thuật toán.

Nhận sự kiện nhập và deltatime để cập nhật trạng thái Game:

```
updateState(input, deltaTime);
```

Nhận đối tượng Graphics và Audio để trình diễn, xuất ra phần cứng thiết bị.

```
present(graphics, audio, deltaTime);
```

Khi người chơi thoát Game, tiến hành giải phóng tài nguyên.

```
cleanupResources();
```

Game Framework cần một giao diện để thực hiện các việc sau:

- Thiết lập cửa sổ và thành phần giao diện người dùng (UI) và đăng ký với hệ điều hành, từ đó có thể nhận sự kiện nhập.
- Khởi động Main Loop
- Theo dõi màn hình hiện tại và thực hiện update, present chính nó trong mỗi vòng lặp.
- Chuyển đổi các sự kiện cửa sổ (resume và pause) từ tiến trình giao diện người dùng để Main Loop có thể chuyển chúng từ màn hình hiện tại, có thể thay đổi trạng thái cho phù hợp.
- Điều khiển truy cập đến các modules đã thiết kế: File I/O, Graphics, Input, Audio.

Sau khi thiết kế Game Framework, chúng ta chỉ quan tâm đến xây dựng Game mà không cần bận tâm vòng lặp chính đang làm gì, khi nào chúng ta cần đồng bộ với

tiếng trình giao diện người dùng hay không. Chỉ cần quan tâm đến thực thi giao diện Game với sự trợ giúp từ các modules level thấp và các cảnh báo từ sự kiện cửa sổ. Chúng ta thiết kế một giao diện Game, ẩn hết các chi tiết phức tạp và một lớp abstract Screen để extends tất cả các màn hình.

```
public interface Game {
    public Input getInput();
    public FileIO getFileIO();
    public Graphics getGraphics();
    public Audio getAudio();
    public void setScreen(Screen screen);
    public Screen getCurrentScreen();
    public Screen getStartScreen();
    public void close();
}
```

Tạo thể hiện của các modules cấp thấp phía dưới.

```
Input getInput(); FileIO getFileIO(); Graphics
getGraphics(); Audio getAudio();
```

Thiết lập màn hình cần vẽ, Main Loop sẽ yêu cầu màn hình hiện tại update và present chính nó.

```
void setScreen(Screen screen);
```

Phương thức trả về màn hình đang kích hoạt.

```
Screen getCurrentScreen();
```

Trả về một thể hiện của màn hình đầu tiên được kích hoạt, là điểm khởi đầu đề chuyển sang các màn hình khác mà ta sẽ thực thi ở phần sau.

```
Screen getStartScreen();
```

Phương thức thoát Game.

```
void close();
public abstract class Screen {
    protected final Game game;
```

```
        public Screen(Game game) {
             this.game = game;
        }
        public abstract void update(float deltaTime);
        public abstract void present(float deltaTime);
        public abstract void pause();
        public abstract void resume();
        public abstract void dispose();
    }
```

Bộ khởi dựng sẽ nhận được 1 thể hiện của Game, lưu trữ nó trong đối tượng final có thể truy cập vào các lớp con. Thông qua cơ chế này ta đạt được 2 điều:

- Dễ dàng truy cập xuống các modules ở mức thấp hơn để chơi nhạc, vẽ lên màn hình, truy xuất file, nhận dữ liệu người dùng nhập vào.
- Có thể chuyển màn hình cần vẽ qua phương thức Game.setScreen().

```
void update(float deltaTime);
void present(float deltaTime);
```

Hai phương thức này giúp Game có thể tự cập nhật, vẽ chính nó trong Main Loop. Hai phương thức này gọi khi Game tạm dừng hoặc trở lại.

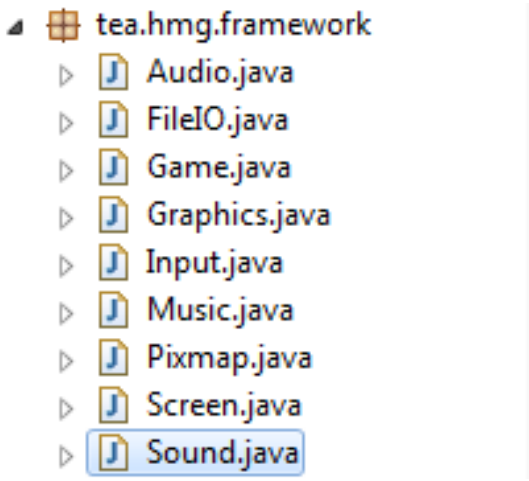
```
void pause();
void resume();
```

Chỉ được gọi sau khi `Game.setScreen()` đã gọi. Phương thức giải phóng tài nguyên Game sau khi thoát.

```
void dispose();
```

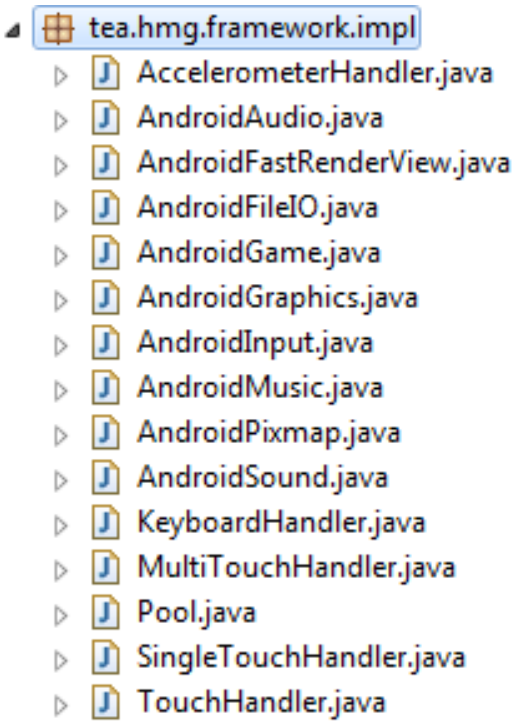
2.2.7 Thực thi Game Framework.

Với bộ giao diện đã thiết kế, cần tiến hành thực thi giao diện dưới nền tảng API của Android. Các giao diện đã thiết kế cần được thực thi:



Hình 8: Các giao diện

- Thiết kế lớp AndroidAudio thực thi giao diện Audio.
- Thiết kế lớp AndroidInput thực thi giao diện File I/O.
- Thiết kế lớp AndroidGame thực thi giao diện Game.
- Thiết kế lớp AndroidGraphics thực thi giao diện Graphics
    - Thiết kế lớp AndroidFastRenderView thực hiện việc vẽ lên màn hình. Nắm giữ vòng lặp chính của Game, làm việc với tiến trình đồ họa.
- Thiết kế lớp AndroidSound thực thi giao diện Sound
- Thiết kế lớp AndroidMusic thực thi giao diện Music
- Thiết kế lớp AndroidPixmap thực thi giao diện Pixmap
- Thiết kế các lớp con của AndroidInput để nhận sự kiện từ người dùng. Tương ứng với các loại sự kiện:
    - Cảm biến gia tốc: AcceleronmeterHandler
    - Đơn chạm: SingleTouchHandler
    - Đa chạm: MultiTouchHandler
    - Bấm phím: KeyboarHandler
    - Lớp Pool chứa danh sách các sự kiện nhận từ các lớp con của AndroidInput.
- Thiết kế lớp AndroidGame thực thi giao diện Game



Hình 9: Các lớp thừa kế bộ giao diện bằng API của Android

Các lớp trên sẽ thực thi giao diện đã được thiết kế bằng các API của Android, tại lớp Android Game, ta khởi tạo các đối tượng sau để phục vụ cho Game:

```
AndroidFastRenderView renderView;
Graphics graphics;
Audio audio;
Input input;
FileIO fileIO;
Screen screen;
```

Điểm khởi đầu của một ứng dụng trong Android là phương thức onCreate(). Tại đây các đối tượng được khởi tạo như sau:

```
renderView = new AndroidFastRenderView(this, frameBuffer);
graphics = new AndroidGraphics(getAssets(), frameBuffer);
fileIO = new AndroidFileIO(getAssets());
audio = new AndroidAudio(this);
input = new AndroidInput(this,renderView,scaleX,scaleY);
screen = getStartScreen();
```

Ở đây ta chú ý đến tỉ lệ màn hình, Game sẽ lấy tỉ lệ mặc định là 320x480. Với màn hình lớn hơn, khung hình cần vẽ được chia cho kích thước thật của màn hình để tạo 1 tỉ lệ, giúp căn chỉnh khung hình vừa với màn hình.

Đối tượng cần vẽ lên màn hình được thực thi bởi lớp Canvas trong Android, lớp này thực hiện đơn giản là vẽ một hình ảnh lên màn hình, hình ảnh này được thiết lập bởi nhiều đối tượng và sự cập nhật, vẽ chính đối tượng riêng lẻ tạo nên. Sau khi các đối tượng cập nhật lại trạng thái, framebuffer sẽ vẽ nó lên 1 Bitmap và chờ để vẽ lên màn hình.

Khai báo một Bitmap cho lớp Canvas

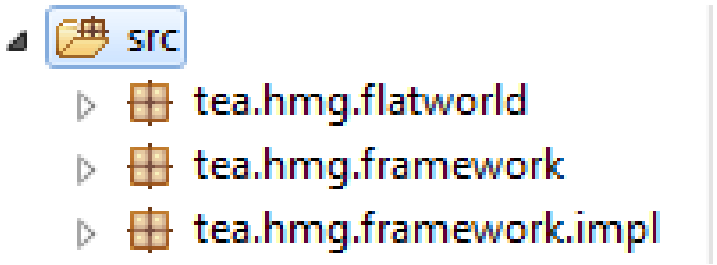
```
Bitmap frameBuffer = Bitmap.createBitmap(frameBufferWidth,
        frameBufferHeight, Config.RGB_565);
```

Tỉ lệ khung hình trục X = độ rộng frameBuffer / độ rộng màn hình.

Tỉ lệ khung hình trục Y = độ dài frameBuffer / độ dài màn hình.

Cuối cùng lớp này dùng phương thức `setContentView(renderView);` để đặt đối tượng renderView bắt đầu vẽ hình ảnh của Game lên màn hình.

Sau khi đã có Framework và thực thi theo API của Android. Để dễ quản lý, ta đóng gói Framework và một gói (package), bộ thực thi vào 1 gói, và phần nội dung Game vào một gói, tên gói được đặt theo quy tắc của Android.



Hình 10: Các gói trong dự án Game

- Gói `tea.hmg.framework` chứa các giao diện.
- Gói `tea.hmg.framework.impl` chứa các lớp thực thi giao diện
- Gói `tea.hmg.flatworld` chứa các lớp của Game

Sau khi thiết kế Game Framework và thực thi bằng API của Android. Công đoạn tiếp theo là Phân tích, Thiết kế Game.

2.3 Phân tích Game

Với phương thức `setScreen()` của lớp Game. Ta có được điểm khởi đầu để vẽ màn hình đầu tiên. Game yêu các lớp đáp ứng các yêu cầu sau:

- Tải tải nguyên lúc khởi động Game vào RAM để sử dụng về sau.
- Mỗi màn hình được thiết kế độc lập, có khả năng tự cập nhật và vẽ lại chính nó khi được kích hoạt bởi phương thức setScreen() của lớp Game.
- Mỗi đối tượng được thiết kế độc lập, có phương thức tự cập nhật trạng thái khi nhận được tương tác truyền vào từ người chơi.
- Một thể hiện của lớp Game được truyền vào mỗi màn hình để kết nối đến các lớp ở mức dưới (Framework) để sử dụng các phương thức được cung cấp.
- Phát âm thanh, nhạc nền.
- Lưu điểm của người chơi và xếp hạng từ cao xuống thấp, lưu các thiết đặt của Game(bật/tắt âm thanh).
- Các tài nguyên cần sắp xếp vào một lớp để dễ dàng truy cập, Game cần thêm một lớp về các tài nguyên (âm thanh, hình ảnh).
- Các thiết lập và điểm số cần được lưu trữ, một lớp đảm nhận việc kiểm soát các thiết lập và điểm số.

Game "FLAT WORLD" được mô tả: Một mặt bàn với nhiều hố đen, có một đích đến và một viên bi. Người chơi sử dụng cảm biến gia tốc, nghiêng điện thoại để lăn viên bi đến đích, tránh các lỗ đen. Như vậy ta xác định Game có các đối tượng sau:

1. Viên bi:
    - Tự cập nhật vị trí khi nhận tín hiệu cảm biến gia tốc và thời gian delta time
    - Xác định va chạm với các hố đen và đích.
    - Xác định va chạm với biên bản đồ để không chạy vượt ra ngoài

- Khởi tạo với một vị trí x, y trên bản đồ được truyền vào.

2. Đích
    - Khởi tạo với vị trí x, y trên bản đồ được truyền vào.
3. Hố đen
    - Khởi tạo với vị trí x,y trên bản đồ được truyền vào.
4. Bản đồ (phân bố vị các hố đen, đích, viên bi)
    - Khởi tạo màn chơi mỗi cấp độ với vị trí hố đen, vị trí đích, vị trí viên bi ngẫu nhiên. Với số lượng xác định theo mỗi cấp độ. Ở đây xác định số viên bi và đích luôn là 1, số hố đen tối thiểu ở cấp 1 là 5.
    - Tạo các đường đi cố định.
    - Chọn ngẫu nhiên một trong số các đường đi cố định, sau đó tạo ngẫu nhiên các hố đen vào các vị trí trống còn lại. Mục đích đảm bảo luôn có đường đi tới đích.
    - Xác định khi nào viên bi va chạm với đích hoặc hố đen để xác định kết quả màn chơi.
    - Tính thời gian cho người chơi, hết thời gian tự kết thúc màn chơi
    - Tính điểm cho người chơi qua từng cấp độ.
    - Cập nhật lại màn hình của chính nó.

Game sẽ chuyển giữa nhiều màn hình. Các màn hình Game đã được xác định như sau:

1. *Màn hình khởi động* (Loading Screen)
2. *Màn hình chính* (Main menu Screen)
    1) *Màn hình thoát Game* (Game quit Screen)
3. *Màn hình hướng dẫn* (Help Screen)
4. *Màn hình xem điểm* (High Score Screen)
5. *Màn hình xem thông tin Game* (About Screen)
6. *Màn hình chơi Game* (Game Screen)
    1) *Màn hình trạng thái Sẵn sàng* (Game Ready)
    2) *Màn hình trạng thái đang chơi game* (Running)
    3) *Màn hình trạng thái qua màn chơi* (Game win)
    4) *Màn hình trạng thái thua màn chơi* (Game Over)

5) *Màn hình trạng thái Tạm dừng Game* (Game Paused)
6) *Màn hình trạng thái Thoát Game* trở về *Mành hình chính* (Game Quit)

Theo phân tích. Mỗi màn hình Game sẽ phân thành một lớp, trong đó *Màn hình Game*(2) có màn hình con là *Màn hình thoát Game*(2.1) sẽ được xử lý trong nội tại lớp, không cần phải tạo lớp mới, tương tự cho *Màn hình chơi Game*(6).

Các lớp về màn hình:

1. *Màn hình khởi động* (Loading Screen)
2. *Màn hình chính* (Main menu Screen)
3. *Màn hình hướng dẫn* (Help Screen)
4. *Màn hình xem điểm* (High Score Screen)
5. *Màn hình xem thông tin Game* (About Screen)
6. *Màn hình chơi Game* (Game Screen)

Phân tích *Màn hình khởi động*: Màn hình tự chuyển sang *Màn hình chính* sau 4 giây để người chơi xem logo của nhà sản xuất, tên trò chơi. Trong thời gian đó màn hình tự vẽ chính nó đồng thời tải các tài nguyên cho Game vào RAM.

Phân tích *Màn hình chính*: Màn hình vẽ các nút bấm chuyển đến các màn hình khác, là gốc để các màn hình khác trở về. Cho phép tắt mở âm thanh, chuyển đến *Màn hình xem điểm, Màn hình thông tin Game, Màn hình chơi Game, Màn hình hướng dẫn, Màn hình con thoát Game.*

Phân tích *Màn hình chơi Game*: Nội dung Game được vẽ tại đây, hiện thị hình ảnh của các đối tượng Viên bi, Hố đen, Đích, điểm, thời gian chơi còn lại, cấp độ và nút dừng (pause Game). *Màn hình chơi Game* chuyển đến các màn hình con sau:

1. *Màn hình sẵn sàng*: Sau khi đã chuẩn bị xong màn chơi bên dưới, màn hình này hiện thị, chờ người chơi chạm vào màn hình sẽ khởi động Game ngay lập tức.
2. *Màn hình thoát Game*: Khi người chơi dừng Game và chọn nút thoát Game, màn hình này sẽ hiển thị, hỏi người chơi có chắc chắn muốn thoát hay không. Thoát thì chuyển đến màn hình xem điểm cao, không thì quay lại màn hình dừng Game.

3. *Màn hình dừng Game*: Khi dừng Game màn hình hiển thị 3 tùy chọn: Thoát Game, chơi tiếp, bật/tắt âm thanh. Khi nhấn chơi tiếp sẽ chuyển về *Màn hình chơi Game*, nhấn thoát sẽ chuyển đến *Màn hình thoát Game*, nhấn tắt/bật âm thanh.
4. *Màn hình thắng Game*: Hiển thị người chơi đã thắng Game, sau 1.5 giây tự chuyển qua màn hình sẵn sàng của màn tiếp theo.
5. *Màn hình thua Game* : Dừng 1.5 giây để thông báo đã thua Game, sau đó tự chuyền về màn hình điểm cao để người chơi xem điểm.

Sau khi phân tích các lớp cần thiết, ta tiến hành thiết kế các lớp với các thuộc tính và phương thức xác định các yêu cầu trên.

2.4 Thiết kế Game

2.4.1 Lớp main (HMG – tên project)

Lớp kế thừa từ AndroidGame, có nhiệm vụ làm điểm khởi đầu cho Game, từ đó chuyển qua màn hình cần vẽ, lớp chỉ kế thừa phương thức `getStartScreen()` sau đó trả về 1 đối tượng của Lớp tài nguyên.

```
public class HMG extends AndroidGame{
     public Screen getStartScreen() {
          return new LoadResource(this);
     }
}
```

2.4.2 Lớp tài nguyên(Assets)

Lớp này định nghĩa các biến tĩnh về tất cả tài nguyên trong Game. Tài nguyên trong 1 dự án phần mềm tạo ra cho Android đặt trong thư mục Assets. Các biến hình ảnh sử dụng kiểu dữ liệu Pixmap, âm thanh sử dụng kiểu Sound, nhạc sử dụng kiểu Music. Các hình ảnh dùng trong Game:

1. Hình ảnh nền Game, khởi động, màn hình chính, giúp đỡ, thông tin.
2. Viên bi, Các Hố đen, Đích,

3. Các nút bấm,
4. Các bảng thông báo: thắng, thua, thoát game, tạm dừng.
5. Âm thanh nền, âm thanh va chạm với hố đen, đích, đếm lùi hết giờ, chạm.

```
public class Assets {
        Khai báo các biến tài nguyên();
        public static Pixmap background;
        ...
        public static Music musicBackground;
        ...
        public static Sound soundClick;
}
```

2.4.3 Lớp tải tài nguyên (LoadResource)

Các tài nguyên sau khi khai báo cần được khởi tạo. Các lớp cần vẽ lên màn hình đều kế thừa từ lớp Screen và được truyền một đối tượng game để truy cập các cấp thấp hơn. Tại lớp này, phương thức update() sẽ khởi tạo các đối tượng khai báo ở lớp Assets.

```
class LoadResource extends Screen {
        Phương thức Khởi tạo(){ };
        void update(float deltaTime) {
                Khởi tạo các đối tượng trong lớp Assets();
                Tải điểm số và các thiết lập();
        }
        void present(float deltaTime) {
                Vẽ hình ảnh đang tải game();
                Chuyển đến màn hình chính();
        }...
}
```

2.4.4 Lớp Màn hình chính (MainScr)

Lớp có 2 trạng thái nội tại là hiển thị màn hình chính và hiển thị màn hình khi người chơi nhấn vào thoát Game. Cần khai báo một kiểu liệt kê cho 2 trạng thái này, sau đó tùy vào sự kiện nhận được mà chuyển màn hình. Lớp kế thừa từ Screen.

```
class MainScr extends Screen {
     enum mainState { Normal, Quit }
     MainScr(Game game) {
          state = mainState.Normal;}
     void update(float deltaTime) {}
     public void present(float deltaTime) {}
}
```

Khai báo kiểu trạng thái.

```
enum mainState {Normal, Quit}
```

Phương thức khởi tạo, đặt trạng thái.

```
     MainScr(Game game) {
          state = mainState.Normal;}
```

Theo dõi các sự kiện chạm màn hình để chuyển tới màn hình tương ứng. Nếu trạng thái là mainState.*Normal* (khi người chơi chưa nhấn vào nút thoát Game) → Phát nhạc nền.Theo dõi sự kiện chạm màn hình. Nếu bấm vào các phím tương ứng với các màn hình khác → chuyển đến màn hình đó qua phương thức game.setScreen(**new** Screen(game)). Nếu bấm vào phím Thoát game → chuyển trạng thái qua mainState.*Quit.* Ngược lại, Nếu trạng thái là mainState.*Quit*. Giải phóng tài nguyên. Thoát Game.

```
void update(float deltaTime) {}
```

Vẽ màn hình chính. Nếu trạng thái là mainState.*Normal* → Vẽ các thành phần của màn hình chính. Nếu trạng thái là mainState.*Quit* → Vẽ màn hình thoát Game.

```
void present(float deltaTime){}
```

Trong phương thức update của lớp Main sẽ đợi sự kiện nhập của người chơi để chuyển đến các màn hình tương ứng trên Menu, nếu nhấn vào nút thoát game sẽ chuyển trạng thái qua Quit để phương thức present() nhận biết vẽ màn hình tương ứng.

2.4.5 Lớp Màn hình hướng dẫn (HelpScr)

Lớp đơn giản chỉ vẽ ảnh hướng dẫn lên màn hình, vẽ nút chuyển ảnh, nút chuyển về màn hình chính. Đợi sự kiện nhấn phím của người chơi để chuyển. Game FLAT WORLD có 3 màn hình giúp đỡ. Do đó có 3 đối tượng màn hình được chuyển lần lượt, sau đó chuyển về màn hình chính.

```
class HelpScr extends Screen {
    Khởi tạo đối tượng();
    void update(float deltaTime){
    Đợi  sự kiện nhập() → chuyển tới màn hình tương ứng;
    }
    void present(float deltaTime){
        Vẽ ảnh();
        Vẽ nút bấm();
    }
```

2.4.6 Lớp Màn hình giới thiệu (AboutScr)

Lớp tương tự lớp màn hình hướng dẫn, chỉ thay hình ảnh. Hình ảnh thông tin về nhà sản xuất, tác giả, liên hệ.

```
class AboutScr extends Screen {
    Khởi tạo đối tượng();
    void update(float deltaTime){
    Đợi sự kiện nhập() → chuyển tới màn hình tương ứng;
    }
    void present(float deltaTime){
        Vẽ ảnh();
        Vẽ nút bấm();}
```

2.4.7 Lớp Màn hình xem điểm (HighScoreScr)

Lớp lấy thông tin điểm đọc từ file, vẽ lên màn hình, tương tự các màn hình hỗ trợ,

```
class HighscoreScr extends Screen {
    Khởi tạo đối tượng();
```

```
void update(float deltaTime){
Đợi sự kiện nhập() → chuyển tới màn hình tương ứng;
}
void present(float deltaTime){
      Vẽ ảnh();
      Vẽ nút bấm();
}
```

2.4.8 Lớp Màn hình chơi Game (GameScr)

Lớp màn hình chơi Game là lớp chính để Game có thể vẽ, cập nhật các đối tượng viên bi, đích, hố đen, cấp độ, thời gian…

```
public class GameScr extends Screen {
     int level;
     int levelmax;
     int score;
     float time;
     Page page;
     enum GameState {
          Ready, Running, Paused, GameOver, Win, Quit}
     GameScr(Game game) {
          super(game);
          page = new Page();}
     void update(float deltaTime) {}
     void present(float deltaTime) {}
}
```

State là một kiểu liệt kê, các phương thức sẽ xử lý và chuyển đổi giữa các trạng thái dựa vào sự kiện xảy ra trong game, sau mỗi vòng lặp game, phương thức update và present sẽ dựa vào sự thay đổi của trạng thái mà cập nhật, vẽ lại màn hình cho phù hợp.

```
enum GameState {
```

```
Ready, Running, Paused, GameOver, Win, Quit}
```

- Biến `level` lưu thông tin cấp độ người chơi đạt được.
- Biến `levelmax` xác định cấp độ tối đa của Game.
- Biến `score` lưu điểm của người chơi tại mỗi lượt.
- Biến `time` làm biến trung gian để tính thời gian.
- Đối tượng `page` là đối tượng bản đồ, đặt tên là Page.

Phương thức khởi tạo `GameScr`, khởi tạo đối tượng và tạo mới bản đồ.

```
public GameScr(Game game) {
    super(game);
    page = new Page();
}
```

Phương thức update() cập nhật lại trạng thái game với trạng thái tương ứng nhận từ biến state. Các trạng thái update() được chuyển tới chờ nhận sự kiện từ người chơi để cập nhật lại màn hình hoặc chuyển sang trạng thái khác.

Phương thức present() vẽ bản đồ và vẽ màn hình tương ứng với trạng thái nhận từ biến state. Các trạng thái present() dựa vào trạng thái để vẽ lại các đối tượng sau khi được cập nhật bởi các trạng thái update().

2.4.9 Lớp Thiết lập (Settings)

Lớp thực hiện việc đọc, ghi file điểm. Lưu thiết lập âm thanh. Sắp xếp điểm từ cao xuống thấp.

```
public class Settings {
    static void load(FileIO files){}
    static void save(FileIO files) {}
    static void addScore(int score) {} }
```

Đọc nội dung file, đưa vào danh sách điểm, lấy thiết lập âm thanh.

```
void load(FileIO files){}
```

Lưu mảng điểm, thiết lập âm thanh vào file.

```
void save(FileIO files) {}
```

Thêm điểm vào danh sách, sắp xếp từ cao xuống thấp.

```
void addScore(int score) {}
```

2.4.10 Lớp viên bi (Ball)

Lớp là thể hiện của một viên bi trong mỗi màn chơi. Có các phương thức tự cập nhật vị trí dựa vào delta time, tín hiệu từ cảm biến gia tốc. Phát hiện va chạm biên và va chạm với các hố đen, đích.

```
public class Ball {
    Ball(float x, float y) { }
    boolean circleCollision(Hole hole) {}
    void boundCollision() {}
    void updatePosition(float sensorX, float sensorY,
    float deltaTime) {}
}
```

Phương thức khởi tạo truyền vào vị trí của viên bi.

```
Ball(float x, float y) {}
```

Phương thức xác định va chạm, khi gọi truyền vào 1 đối tượng hố đen. Trả về kết quả true nếu va chạm và ngược lại.

```
boolean circleCollision(Hole hole) {}
```

Phương thức xác định va chạm với biên, khi phát hiện va chạm, phương thức giữ cho viên bi không ra khỏi biên.

```
void boundCollision() {}
```

Phương thức cập nhật vị trí của viên bi, truyền vào 2 giá trị sensorX và sensorY nhận từ lớp GameScr, thời gian delta time. Phương thức dựa vào sự thay đổi của 3 giá trị truyền vào để cập nhật vị trí viên bi cho phù hợp.

```
void updatePosition(float sensorX, float sensorY, float
deltaTime) {}
```

2.4.11 Lớp hố đen và đích (Hole)

Có thể nhập 2 đối tượng này vào một lớp và phân loại bằng thuộc tính type. Đối tượng đơn giản chỉ cần tọa độ và loại của hố, là hố đen hay đích. Định nghĩa lớp như sau:

```
public class Hole {
    public static final int TYPE_HOLE = 0;
    public static final int TYPE_HOLE1 = 1;
```

```
        public static final int TYPE_HOLE2 = 2;
        public static final int TYPE_GOAL = 3;
        public int x,y,type;
        public Hole(int x, int y, int type){
            this.x=x;
            this.y=y;
            this.type = type;
        }
    }
```

Để các hố đen khi tạo ra trên bản đồ phong phú hơn, Game cần định nghĩa 3 loại hố đen bởi 3 biến tĩnh . Còn đích chỉ có 1 cái và một loại duy nhất. Phương thức khởi tạo đối tượng truyền vào tọa độ và thể loại hố.

2.4.12 Lớp Bản đồ (Page)

Bản đồ được tạo ngẫu nhiên mỗi lần chơi cấp độ mới. Để đơn giản, Game định nghĩa tọa độ bản đồ có chiều rộng 10 đơn vị, chiều cao 15 đơn vị. Tạo thành bản đồ 10x15 = 150 đơn vị.

```
public static final int PAGE_WIDTH = 10;
public static final int PAGE_HEIGHT = 15;
```

Với mỗi hố đen, viên bi, đích có kích thước khi vẽ lên mà hình là 32x32, với bản đồ kích thước 10x15, khi vẽ lên màn hình chỉ cần nhân với tỉ lệ 32 sẽ ra tọa độ thực cần vẽ. Ví dụ., viên bi ở vị trí 2x10 trên bản đồ sẽ có vị trí trên màn hình là (2x32, 10x32) = 64x320 trên màn hình được vẽ. Bản đồ cần lần lượt các biến sau đây:

Biến xác định Game đã thắng hay thua:

```
public boolean gameOver = false;
public boolean gameWin = false;
```

Biến lưu điểm qua mỗi màn chơi.

```
public int score = 0;
```

Biến lưu cấp độ qua mỗi màn chơi.

```
public int level = 1;
```

Biến time dùng để tính deltatime.

```
float time;
```

Biến tính thời gian còn lại của từng màn chơi.

```
int printTime;
```

Biến đếm số hố đen cần vẽ.

```
public int numHoles;
```

Mảng lưu các hố đen cần vẽ.

```
public Hole[] holes;
```

Biến viên bi, kiểu dữ liệu Ball.

```
public Ball ball;
```

Biến tạo ngẫu nhiên sử dụng trong tạo vị trí trên bản đồ.

```
Random random;
```

Hai biến lấy vị trí tạo ngẫu nhiên.

```
private int holeX;
private int holeY;
```

Biến đếm số hố đen đã tạo.

```
private int sloted;
```

Mảng bản đồ là một mảng 2 chiều, xác định vị trí trên bản đồ đã có vật thể nào được tạo hay chưa.

```
boolean maps[][] = new boolean[PAGE_WIDTH][PAGE_HEIGHT];
```

Các mảng đường đi được tạo sẵn với các vị trí true ở đầu và cuối khung bản đồ cho viên bi là đích. Các vị trí true còn lại nối từ đích đến viên bi để bảo đảm luôn có đường đi và các hố đen khi tạo mới không được thả vào đường đi này. Bản đồ cần các phương thức sau:

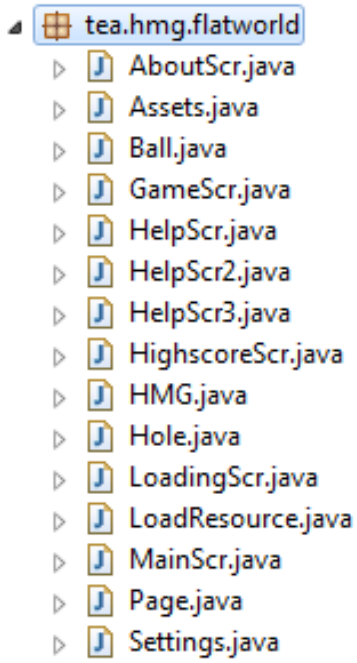
```
public Page() {
     initMaps(level);}
```

`initMaps()` Tạo mới đối tượng Page (bản đồ), khởi tạo bản đồ bằng phương thức initMaps(level) và truyền cấp độ vào để phương thức tạo số hố đen thích hợp. Lấy ngẫu nhiên 1 trong các đường đi được định sẵn, tạo đích, viên bi theo đường đi này. Tạo các hố đen không nằm trong đường đi, bảo đảm bản đồ luôn có đường đi mà không bị các hố đen tạo ngẫu nhiên lấp hết. Tiếp theo là phương thức cập nhật:

```
void update(float sensorX,float sensorY,float deltaTime){}
```

Nhận giá trị cảm biến gia tốc từ lớp GameScr truyền vào cho lớp viên bi để viên bi tự cập nhật lại tọa độ, kiểm tra va chạm với thành để giữ bi luôn trong màn hình, kiểm tra va chạm với hố đen, đích để xác định thắng hay thua Game. Cập nhật thời gian chơi cho từng màn chơi.

Tổng hợp lại, Game được xây dựng với các lớp như sau:



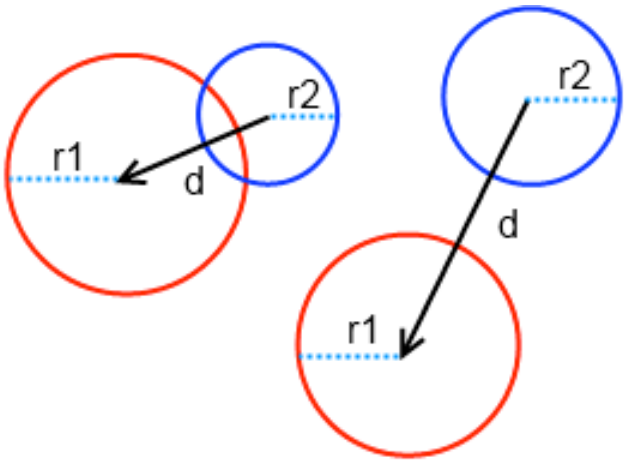
Hình 11: Các lớp cần thiết của Game

2.5 Thiết kế thuật toán

Sau khi thiết kế các lớp và các phương thức. Game cần thuật toán để xử lý các yêu cầu của phương thức đặt ra, các lớp, phương thức chính có sử dụng thuật toán sau:

- Thuật toán xác định viên bi và hố đen va chạm với nhau.
- Thuật toán xác định va chạm với biên và đặt lại vị trí.
- Thuật toán cập nhật vị trí viên bi.
- Thuật toán tạo bản đồ.
- Thuật toán của vòng lặp chính Main Loop.

2.5.1 Thuật toán xác định va chạm giữa viên bi và hố đen hoặc đích.



Hình 12: Minh họa về va chạm giữa 2 hình tròn

Trước khi xây dựng thuật toán, ta cần xây dựng 1 số công thức sau: Viên bi và hố đen là 2 hình tròn, 2 hình tròn va chạm khi khoảng cách giữa tâm của 2 đường tròn nhỏ hơn tổng bán kính 2 đường tròn:

$$d \leq r1 + r2$$

Để tính khoảng cách giữa 2 điểm trong mặt phẳng (d), ta có công thức:

$$\mathrm{d} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

để giảm tính phức tạp, ta bỏ phép căn

$$d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

Thuật toán như sau:

```
boolean circleCollision(Hole hole) {
            distanceX = ball.x – hole.x;
            distanceY = ball.y – hole.y;
            distance = distanceX * distanceX + distanceY *
            distanceY;
            radiusSum = 16;
            if (distance <= radiusSum * radiusSum)
                 return true;
```

```
            else
                return false;
        }
```

Cài đặt thuật toán phần xác định va chạm. `this.x, this.y` là tọa độ của viên bi. `Hole.X`, `hole.Y` là tọa độ của hố đen được truyền vào. `distance` chính là công thức $(x_1 - x_2)^2 + (y_1 - y_2)^2$. Ta cần 1 biến `radiusSum` để xác định khoảng cách va chạm. Trong Game này, khi tâm viên bi cách tâm hố đen một khoảng bằng bán kính (16px) thì xác định là va chạm. Sau đó thực hiện phép so sánh. Nếu khoảng cách nhỏ hơn tổng bán kính hố đen là viên bi thì trả về `true`, ngược lại `false`.

2.5.2 Thuật toán xác định va chạm với biên và đặt lại vị trí

Thuật toán xác định khi nào tọa độ x, y của viên bi đạt tới giới trên hoặc giới hạn dưới của màn hình và đặt lại vị trí ngay vị trí viên bi vừa va chạm với biên. Thuật toán được trình bày: với x, y là tọa độ viên bi trong màn hình. xmax, ymax là tọa độ tối đa viên bi có thể di chuyển tới, tọa độ dưới của màn hình là x =0, y = 0.

Nếu x > xmax thì đặt x = xmax;

Nếu y > ymax thì đặt x = ymax;

Phía biên trên tương tự:

Nếu x < 0 thì x = 0;

Nếu y < 0 thì y = 0;

Như vậy, khi viên bi ra đến biên, trường hợp trên sẽ được xét đến và đặt lại vị trí. Thuật toán được cài đặt trong Game:

```
    public void boundCollision() {
        xmax = screenWidth;// kích thước màn hình theo trục x
        ymax = screenHeigh;// thước màn hình theo trục y
        x = mPosX;// tọa độ x của viên bi
        y = mPosY;// tọa độ y của viên bi
        if (x > xmax)
            mPosX = xmax;
        else if (x < 0)
```

```
            mPosX = 0;
        if (y > ymax)
            mPosY = ymax;
        else if (y < 0)
            mPosY = 0;
    }
```

2.5.3 Thuật toán cập nhật vị trí viên bi

Đây là thuật toán sau cùng sử dụng delta time để cập nhật chuyển động. Thuật toán sử dụng các biến sau:

`timeInit:` Thời gian cho 1 lần cập nhật lại vị trí viên bi. Khởi tạo: 0.004f;

`lastTime:` Giá trị trung gian lưu thời gian cộng dồn giữa các chu kỳ cập nhật.

`mOneMinusFriction:` Hệ số ma sát ảo với mặt bàn.

`mAccelX, mAccelY:` Giá trị cảm biến gia tốc truyền vào.

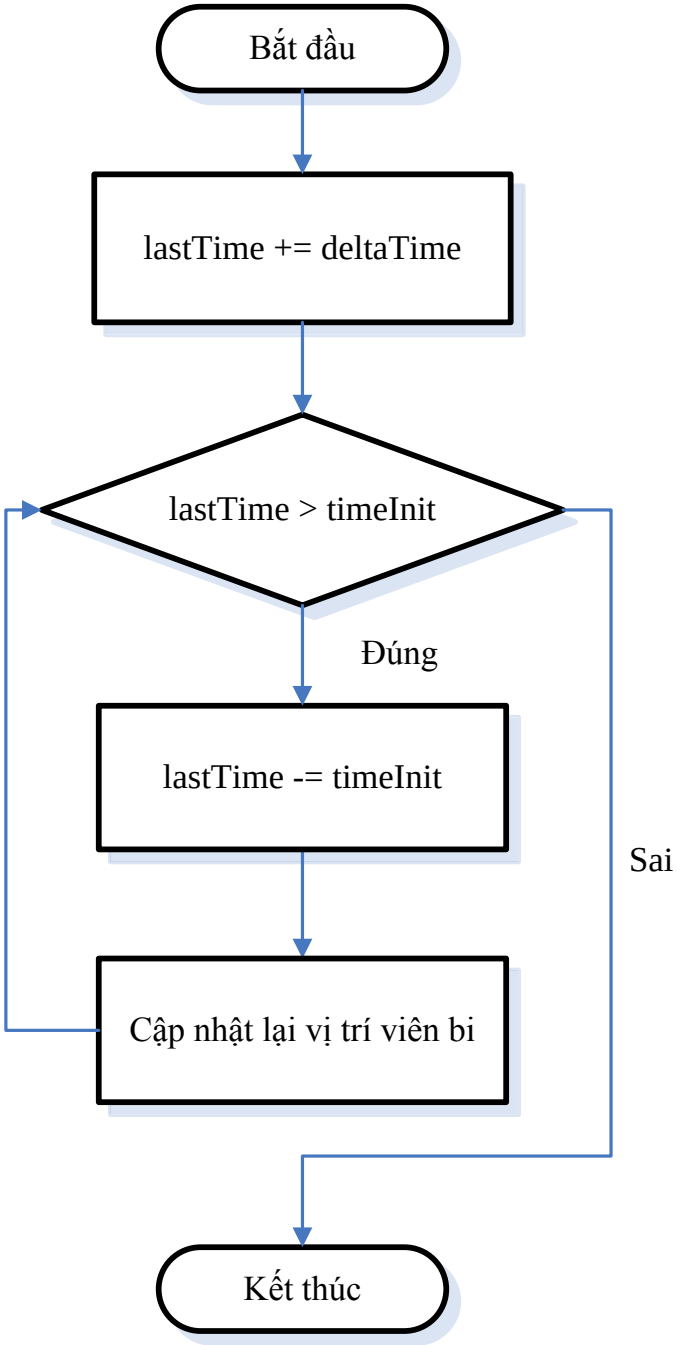
delta time là lượng thời gian trôi qua 1 vòng lặp của Game. Thời gian trôi qua mỗi vòng lặp có thể khác nhau do các tính toán mỗi vòng lặp khác nhau (tính toán va chạm, vẽ hình ảnh). Do vậy, phải thực hiện chuyển động cho viên bi hết số thời gian delta time đã trôi qua. Thuật toán sau với một thời gian delta time bất kỳ, viên bi luôn được cập nhật vị trí mỗi lượng thời gian bằng timeInit cho đến khi hết lượng delta time của vòng lặp.

Để cập nhật vị trí viên bi dựa vào tham số được truyền từ cảm biến gia tốc. Nếu tham số cảm biến x dương thì viên bi di chuyển qua trái, ngược lại thì qua phải. Tương tự với trục y là nghiêng về phía trước thì y dương, nghiêng về phía sau y âm. Để thay đổi vị trí hiện tại dựa vào hướng của thiết bị. Ta tính toán như sau:

Vị trí mới = Vị trí hiện tại + hệ số ma sát * thời gian * (vị trí hiện tại – vị trí trước đó) + giá trị cảm biến gia tốc. Lần lượt tính vị trí `x, y` mới:

```
x = mPosX + mOneMinusFriction * deltaTime * (mPosX -
mLastPosX) + mAccelX;

y = mPosY + mOneMinusFriction * deltaTime * (mPosY -
mLastPosY) + mAccelY;
```



Hình 13 : Thuật toán cập nhật vị trí viên bi

2.5.4 Thuật toán tạo bản đồ.

Để đảm bảo luôn có đường đi đến đích trong khi các hố đen được tạo với vị trí ngẫu nhiên, ta phải vẽ 1 đường đi cố định từ viên bi đến đích tại thời điểm tạo bản đồ, các ô đường đi sẽ được đặt giá trị true trong mảng boolean của bản đồ. Để bản đồ phong phú, Game sẽ được thiết kế sẵn nhiều mảng đường đi, sau đó chọn ngẫu nhiên một mảng cho mỗi màn chơi, mảng đường đi có cấu trúc:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	{ false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false	}
2	{ false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false	}
3	{ false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false	}
4	{ false,	false,	false,	false,	false,	false,	false,	true,	true,	false,	false,	false,	false,	false,	false	}
5	{ false,	false,	false,	false,	false,	true,	true,	false,	false,	true,	false,	false,	false,	false,	false	}
6	{ false,	false,	false,	false,	true,	false,	false,	false,	false,	false,	true,	false,	false,	false,	false	}
7	{ false,	false,	false,	true,	false,	false,	false,	false,	false,	false,	false,	true,	false,	false,	false	}
8	{ false,	false,	true,	false,	false,	false,	false,	false,	false,	false,	false,	false,	true,	false,	true	}
9	{ false,	true,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	true,	false	}
10	{ true,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false,	false	}

Hình 14: Minh họa bản đồ

Đường đi sẽ được thiết lập giá trị true, theo cấu trúc của mảng, hàng ngang là số cột x, hàng dọc là số hàng y trong tọa độ màn hình. Như vậy 2 đầu mang giá trị true là đích (y = 1), viên bi (y = 15). Hàng đầu tiên trong bản đồ được dùng để vẽ điểm số, thời gian, cấp độ nên sẽ không dùng, vì vậy đích được đặt tại hàng y = 1 thay vì y = 0.

Như trong hình (đường đi được đánh dấu màu đỏ), nếu có hố đen được tạo ra sẽ chừa các vị trí này. Tại x = 10, y = 1 sẽ là đích, tại x = 8, y = 15 là vị trí viên bi.

Chọn ngẫu nhiên 1 đường đi trong số các mảng đường đi đã tạo:

```
mapNumber = random.nextInt(10);
        switch (mapNumber) {
        case 1:
              maps = maps1;
              break;
        case 2:
              maps = maps2;
              break;
        ...
        default:
```

```
                maps = maps0;
                break;
            }
```

Khởi tạo bản đồ thiết lập số hố đen bằng cấp độ + 5. Ví dụ., cấp 3 có 3 + 5 = 8 hố đen, gán vào biến `numHoles`.

```
numHoles = level + 5;
```

Tạo mảng các hố đen với số phần từ đã xác định:

```
holes = new Hole[numHoles];
```

Đếm số phần tử đã được tạo bởi 1 biến đếm:

```
sloted = 0;
```

Khởi tạo thời gian chơi mỗi màn bằng cấp độ + 1, 3 cấp đầu có thời gian như nhau:

```
printTime = level+1;
     if(printTime ==1 || printTime == 2 || printTime ==3)
          printTime = 4;
```

Duyệt mảng từ vị trí x = 0 đến x < chiều rộng, tại hàng cuối cùng (y = 15). Tìm thấy phần tử của mảng có giá trị true thì khởi tạo đối tượng viên bi.

```
for (int i = 0; i < PAGE_WIDTH; i++)
     if (maps[i][PAGE_HEIGHT - 1] == true) {
          ball = new Ball(i, PAGE_HEIGHT-1);
          break;
     }
```

Tương tự với đích, nhưng thay cột y bằng 1.

```
for (int i = 0; i < PAGE_WIDTH; i++)
     if (maps[i][1] == true) {
          holes[sloted++]=new Hole(i, 1, Hole.TYPE_GOAL);
          break;
     }
```

Ở đây ta sử dụng mảng `holes[]` để lưu trữ danh sách đối tượng hố đen và đích (có thuộc tính `type` để xác định loại). mảng `holes[0]` vị trí 0 sẽ là đích.

Tiếp theo ta đặt `holes[1]` và `holes[2]` là 2 hố đen ở cạnh trái (x = 0) và cạnh phải (x =10), với y ngẫu nhiên để tránh trường hợp người chơi tận dụng cạnh trái và phải để lăn thẳng tới đích.

```
holeX = random.nextInt(PAGE_WIDTH -1);
holeY = random.nextInt(PAGE_HEIGHT - 1);
```

Tạo ngẫu nhiên 2 biến holeX và holeY để đặt vị trí ngẫu nhiên cho các hố đen. Vị trí y chạy từ 1 (chừa hàng trên cùng để vẽ điểm) và tránh vị trí của đích:

```
if (holeY == 0 || holeY == holes[0].y)
    holeY += 1;
```

Sau đó khởi tạo đối tượng hố đen, thêm vào danh sách, tăng biến đếm lên 1 và đặt vị trí vừa tạo trên bản đồ bằng true để các hố đen tạo sau không bị trùng lại:

```
holes[sloted++] = new Hole(0, holeY, random.nextInt(2));
maps[0][holeY] = true;
```

Tương tự với cạnh phải:

```
holes[sloted++] = new Hole (PAGE_WIDTH-1, holeY,
Random.nextInt(2));
maps[PAGE_WIDTH - 1][holeY] = true;
```

Sau khi thiết lập các thành phần cơ bản của bản đồ, ta đến thuật toán chính tạo ngẫu nhiên các hố đen. Mảng bản đồ với các vị trí true, false tương ứng với có hoặc không có hố đen đã đặt vào vị trí này. Ta duyệt mảng bản đồ, tìm những vị trí chưa có hố đen hoặc đích lưu hết vào mảng là positionList. Mỗi vị trí được mô tả bằng một đối tượng vị trí:

```
class position{
    int x;
    int y;
    position (int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

Thêm các phần từ vị trí vào mảng `positionList` với kiểu dữ liệu `ArrayList` trong Java. Sau khi lưu được một mảng các vị trí trống trên bản đồ, ta chạy 1 vòng lặp nhằm lấy ngẫu nhiên 1 phần từ trong danh sách vị trí, đặt hố đen lên bản đồ với vị trí vừa lấy, sau đó loại phần tử này ra khỏi mảng, rồi lặp lại cho đến khi đủ số hố đen.

Phương thức `random` trong `Java` tạo ra các số từ 0 đến giá trị truyền vào phương thức. Mỗi lần ta lấy ra 1 số, sau đó xóa khỏi mảng, mảng sẽ được sắp xếp lại. Nên vòng lặp tiếp theo chỉ cần lấy 1 số ngẫu nhiên từ 0 – `size` của mảng mà không bị lặp lại. Ta có thuật toán như sau:

```
//Lấy các vị trí trống vào mảng:
Duyet_mang banDo[i][j]{
     if(banDo[i][j] == false)
          new position[i][j];
          positionList.add(position[i][j]);
}
//Duyệt mảng, tạo các hố đen
while (sloted < numHoles) {

          ran = random.nextInt(positionList.size());
          holes[sloted].x = positionList(ran).x;
          holes[sloted].y = positionList(ran).y;
          positionList.remove(ran);
     }
//Xóa hết các phần tử trong mảng để phục vụ lần tạo sau.
positionList.clear();
```

2.5.5 Thuật toán của vòng lặp chính (MainLoop)

Lớp AndroidGame là điểm khởi đầu của ứng dụng sẽ khởi tạo một đối tượng AndroidFastRenderView để vẽ tất cả mọi thứ lên màn hình, vì vậy, vòng lặp Game sẽ được đặt trong lớp AndroidFastRenderView. Với phương thức run()

```
public void run() {
     thoiGianBatDau = thoiGianHienTai;
```

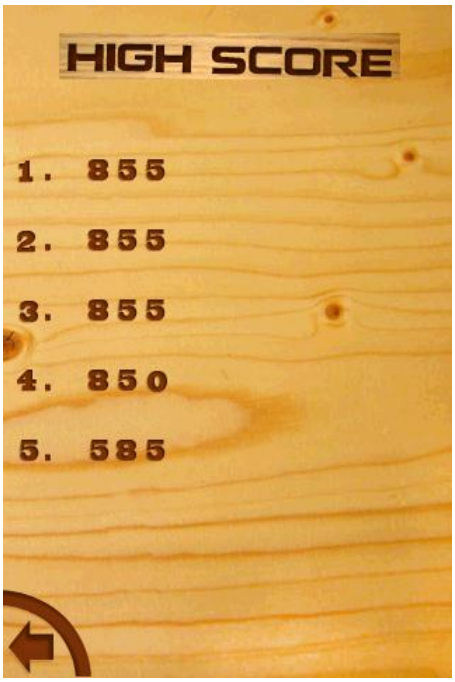
```
        while (running) {
            deltaTime = (thoiGianHienTai – thoiGianBatDau);
            thoiGianBatDau = thoiGianHienTai;
            capNhatManHinh(deltaTime);
            veManHinh(deltaTime);
        }
    }
```

Vòng lặp chính sẽ chạy cho đến khi `runing = false`, tức là khi người chơi thoát Game, vòng lặp tính thời gian bắt đầu, sau đó tính thời gian trôi qua giữa 2 vòng lặp để truyền cho phương thức cập nhật, vẽ của màn hình hiện tại. Cuối cùng vẽ khung hình mới cập nhật lên màn hình. Các đối tượng bên dưới được truyền deltaTime để tính toán chuyển động.

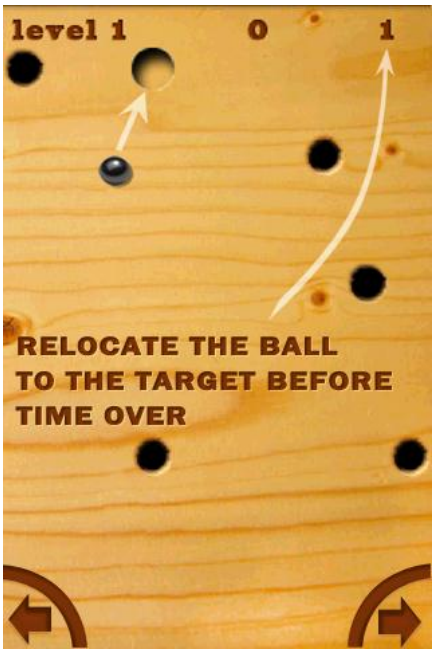
2.6 Thiết kế giao diện

Hình 15: Giao diện chính

Hình 16: Giao diện chơi Game



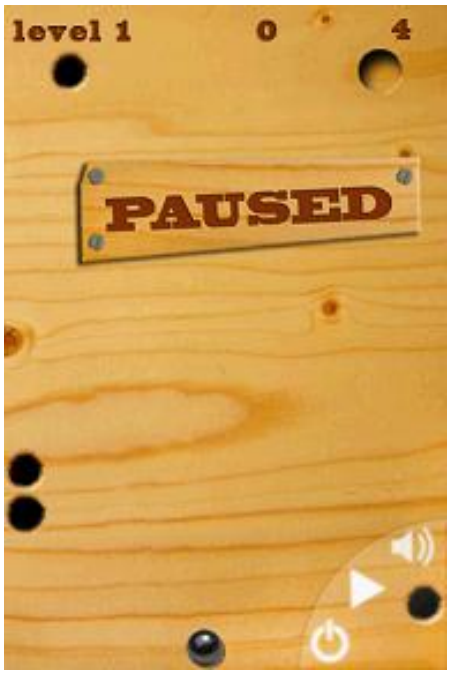
Hình 17: Giao diện xem điểm



Hình 18: Giao diện giúp đỡ

Hình 19: Giao diện thoát Game

Hình 20: Giao diện thoát màn chơi



Hình 21: Giao diện dừng game

Hình 22: Giao diện thua Game

Kết luận: Sau khi phân tích thiết kế Game, ta tiến hành xây dựng Framework, Framework đã xử lý các công việc ở mức thấp nên chỉ cần thực thi giao diện Framework vào Game. Xây dựng các thuật toán xử lý trong Game, xây dựng giaod iện, xử lý các vấn đề liên quan. Cuối cùng chạy Game và kiểm tra lỗi, tối ưu code, hoàn thành Game.

CHƯƠNG III: CHƯƠNG TRÌNH DEMO

3.1 Cài đặt, chạy các chức năng của chương trình demo.

Game được tiến hành cài đặt trên điện thoại LG Optimus Black chạy hệ điều hành Android 2.2, điện thoại Sony Xperia mini chạy Android 2.3.4.

Hình 23: Điện thoại LG LP970 (trái) và Xperia mini (phải)

Sau khi được biên dịch, file cài đặt của chương trình nằm ở thư mục /bin trong thư mục project của Eclipse, copy file này vào thẻ nhớ và cài đặt trên điện thoại chạy Android, Game hỗ trợ phiên bản Android 1.5 trở lên. Sau khi cài đặt, icon của trò chơi sẽ nằm trong menu chương trình của điện thoại, click vào để bắt đầu.

3.2 Hướng dẫn sử dụng.

Game được thiết kế trực quan, một màn hình chỉ có 1 - 3 tùy chọn như thoát game (yes - no), bật - tắt âm thanh... Người chơi sẽ dễ dàng điều khiển Game sau 1 thời gian ngắn để làm quen.

Hình 24: Icon của game trong màn hình Menu của điện thoại

Hình 25: Giao diện chơi Game trên điện thoại

KẾT LUẬN

1. Kết quả đạt được

Qua quá trình thực hiện đồ án chuyên đề Game FLAT WORLD trên hệ điều hành Android với sự hướng dẫn của thầy Nguyễn Thanh Cẩm, em đã cài đặt thành công Game FLAT WORLD chạy trên hệ điều hành Android 1.5, chạy thử trên thiết bị LG LP970 và Sony Xperia mini ổn định. Em cũng nắm được cơ sở lý thuyết để viết Game, có thêm kinh nghiệm trong việc áp dụng kiến thức đã học vào thực tế và học hỏi công nghệ mới. Đồ án sẽ tạo tiền đề cho việc nghiên cứu phát triển các Game sau này. Sản phẩm có thể đưa vào thực tế dễ dàng, qua đó thấy được tính thực tế của đề tài và mở đầu cho các hướng nghiên cứu sử dụng công nghệ mới cho sinh viên về sau.

2. Hướng mở rộng

Game được làm bằng nền tảng đồ họa 2D cơ bản, đơn thuần sử dụng lớp Canvas của Android để vẽ các ảnh lên màn hình, để đồ họa tốt hơn cần chuyển qua nền tảng OpenGL với đồ họa 3D, có chiều sâu hình ảnh và chuyên nghiệp hơn, hay được dùng để phát triển Game trong thực tế. Khi mở rộng, có thể thêm các item giúp tăng thời gian chơi, giảm các hố đen, ăn điểm được tạo trên màn hình, thiết kế đường đi phức tạp hơn, sử dụng âm thanh sinh động hơn.

Đồ án có thể mở rộng các chi tiết trên thành đồ án tốt nghiệp hoặc tối ưu code, trau chuốt lại hình ảnh để có thể tham gia chợ ứng dụng Anroid, người chơi dễ dàng tải về và sử dụng.

HẠN CHẾ

Hạn chế lớn nhất là về đồ họa Game và các hiệu ứng chuyển động còn thô sơ, chuyển động của viên bi chưa giống với thực tế như thiếu lực đàn hồi, gia tốc còn chậm. Âm thanh chưa được phong phú. Thời gian phát triển sản phẩm ngắn nên một số đoạn code chưa được tối ưu cho tốc độ chạy Game, thiết kế bản đồ chưa phong phú.

TÀI LIỆU THAM KHẢO

[1] **Mario Zechner** (2010) *Apress - Beginning Android Games*

[2] Website : *http://developer.android.com/*

[3] Website : *http://java2s.com/*

[4]Website : *http://stackoverflow.com/*

NHẬN XÉT CỦA CÁN BỘ HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Đà Nẵng, ngày ….... tháng …… năm ……

Cán bộ hướng dẫn

Nguyễn Thanh Cẩm

NHẬN XÉT CỦA CÁN BỘ PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Đà Nẵng, ngày ….. tháng ….. năm …...

Cán bộ phản biện