Report cá nhân:

* Các task được giao và tiến độ hoàn thành:
  + Framework
  + Camera
  + Particle
  + Hiện thực phần cơ bản của những state trong game WOT
  + Xửa lý tương tác người dùng với game WOT
  + Định kì port lên android kiểm tra bug
* Nội dung chi tiết từng task

Framework [Đa phần sử dụng lại code mẫu của anh Nguyễn Khánh Duy, trừ các phần như CGLES2Driver, CGraphic, Camera, 2 lớp View, các lớp quản lý resource]

* Quản lý vòng đời game – class CGame: lớp đầu não để quản lý một game, gồm các phương thức chính sau:
  + Init: thiết lập các tham số của game, qui định state bắt đầu của game
  + Destroy: hủy game, gọi trước khi kết thúc, dùng để thu hồi các vùng nhớ, giải phóng thiết bị chiếm giữ
  + Exit: yêu cầu kết thúc game
  + Update: do hiện thực lớp CGame cho mục đích đa nền tảng, vòng lặp chính của game sẽ do lớp View của từng nền tảng quản lý, CGame chỉ có phương thức Update để xử lý ở mỗi vòng lặp chứ không quản lý cả vòng lặp
  + Pause: tạm dừng game
  + Resume: khôi phục game sau khi tạm dừng
* Quản lý các state của game – class CState, CStateManagement:
  + Ứng dụng chia để trị, nên chia game thành nhiều state để quản lý hiệu quả hơn. Ta hiện thực lớp CState
  + Mỗi CState gồm 4 phương thức cơ bản:
    - Init: thiết lập thông số cho mỗi state
    - Update: Xử lý logic ở mỗi state. Quá trình này bao gồm tất các mọi việc xử lý của state ngoại trừ việc vẽ ra màn hình. Việc tách biệt thao tác vẽ ra khỏi các thao tác khác giúp người lập trình thuận lợi hơn trong việc viết code.
    - Render: Thao tác vẽ ra màn hình
    - Exit: Kết thúc state
  + Để đảm bảo việc chuyển đổi giữa các state được trơn tru, ta cần lớp CStateManagement. Việc quản lý state tuân theo nguyên tắc sau:
    - Tại một thời điểm, chỉ có 1 state được phép "hoạt động" (Update/Render)
    - Khi chuyển từ một state (A) sang một state khác (B), A phải được hủy (Exit) và B phải được tạo (Init) sau đó
    - Chỉ chuyển state (chuyển sang state khác) khi State cũ đã kết thúc việc update & render.
  + Hiện thực đươc các nguyên tắc trên, CStateManagement gồm 2 phương thức chính:
    - Update: Kiểm tra xem có cần phải chuyển sang state khác. Nếu phải chuyển state, gọi phương thức Exit của state cũ, gọi phương thức Init của state mới. Sau quá trình kiểm tra chuyển state, gọi phương thức Update của state hiện tại. Việc này đảm bảo 3 nguyên tắc đã nêu ở trên.
    - SwitchState: Đánh dấu việc chuyển state, để khi lần tiếp theo CStateManagement gọi phương thức Update thì sẽ chuyển qua state đã đánh dấu.
* Quản lý FPS – class CFpsController:

Một lần lặp trong vòng lặp game được gọi là frame. Mỗi frame sẽ thực hiện 2 quá trình chính là Update và Render. Tùy thời điểm, tùy thiết bị, việc xử lý của mỗi frame sẽ tốn thời gian thực hiện khác nhau. Do đó game có thể có tốc độ nhanh chậm khác nhau ở những thời điểm khác nhau, những thiết bị khác nhau. Để đồng bộ tốc độ đó, đòi hỏi một lớp quản lý FPS của game. FPS là khái niệm chỉ số frame thực hiện được trong một giây.

Kỹ thuật giới hạn FPS:

* Ước lượng thời gian nhiều nhất để thực hiện một frame, gọi thời gian đó là T.
* Ở tại một frame bất kì, giả sử frame đó cần tốn một thời gian t để thực hiện (t<T), game sẽ gọi một phương thức để “ngủ” trong khoảng thời gian còn lại, tức là T-t. Do đó, tất cả mọi frame đều tốn cùng một khoảng thời gian để thực hiện là T.
* Tuy nhiên, việc giới hạn FPS sẽ không hiệu quả nếu người viết game ước lượng sai T, khi đó sẽ tồn tại những frame mà việc thực hiện vượt quá thời gian T, khiến game nhanh chậm bất thường. Giới hạn FPS càng cao, game càng mượt, tuy nhiên càng dễ bị vượt ước lượng thời gian ở những frame xử lý nhiều. Ngược lại, giới hạn FPS càng thấp, game càng an toàn, nhưng đồ họa không mượt mà. Thông thường, một game được gọi là mượt nên được giới hạn ở mức 25 fps , tức là T=40ms/frame. Mức FPS tối thiểu là 8 fps.

Hiện thực:

* Tạo class CFpsController, gồm 5 phương thức chính là:
  + SetLimitFps: giới hạn mức FPS của game.
  + BeginCounter: gọi hàm kiểm tra thời gian hệ thống, đánh dấu thời gian hiện tại khi gọi hàm.
  + EndCounter: kiểm tra thời gian đã tiêu tốn từ khi gọi BeginCounter đến hiện tại, gọi hàm “ngủ” trong khoảng thời gian còn lại so với FPS đã giới hạn.
  + GetFrameDt: lấy thời gian xử lý frame hiện tại.
  + GetRuntimeFps: lấy FPS hiện tại, đo đạc thực tế, có thể khác với FPS đã giới hạn.
* BeginCounter phải được gọi ở đầu mỗi vòng lặp game, EndCounter ở cuối vòng lặp, để đảm bảo mỗi frame tiêu tốn đúng lượng thời gian đã giới hạn
* View cho win [viết bởi Quân, bảo trì nâng cấp bởi Tín] và android hỗ trợ GLES2
  + CVSView: nhận thông tin về cửa sổ hỗ trợ các hàm OpenGL ES 2 được tạo ra từ việc sử dụng các hàm được hỗ trợ từ thư viện ESUtil.
  + CAndroidView: nhận thông tin về lớp GLSurfaceView được tạo ra từ code Java.
  + 2 lớp View này thay lớp CGame giữ vòng lặp chính, liên tục kiểm tra các tương tác người dùng (với cửa sổ Windows/SurfaceView), xử lý và gửi vào lớp CGame
* Quản lý tương tác người dùng – tương tác trực tiếp (method OnControlerEvent) và gián tiếp (CControllerEventManager):

View nhận tương tác từ người dùng, xử lý và chuyển thông tin đến CGame.

CGame nhận thông tin về tương tác, phân thành 2 luồng xử lý chính:

* Xử lý gián tiếp: chuyển thông tin qua lớp CControllerEventManager và đưa vào hàng đợi tương ứng tùy loại tương tác. Việc sử dụng được thực hiện ở hàm Update, thông qua lớp CControllerEventManager truy xuất hàng đợi bằng các hàm tương ứng của 2 lớp con:
  + CControllerPointerManager: chuyên cho thu lập, xử lý, và cung cấp các hàm tiện ích cho xử lý touch. Các hàm tiện ích cơ bản bao gồm:
    - GetActivePointer: lấy danh sách các điểm touch hiện tại
    - WasTouchedInside / WasReleaseInside: kiểm tra xem người dùng có đang touch hoặc release tại một vùng nào đó hay không
  + CControllerKeyManager: chuyên cho quản lý key. Các hàm tiện ích:
    - WasKeyPressed / WasKeyHold / WasKeyRelease: kiểm tra trạng thái của 1 phím nào đó
    - WasAnyKeyPressed / WasAnyKeyHold / WasAnyKeyRelease: kiểm tra trang thái của mọi phím
* Xử lý trực tiếp: CGame chuyển thẳng thông tin tương tác đến state hiện hành.
* Việc sử dụng cách xử lý nào tùy thuộc vào người lập trình và mục đích của thao tác cần xử lý. Xử lý gián tiếp giúp quản lý hiệu quả hơn các tương tác, có thể kiểm tra các tương tác phức tạp như nhiều nút cùng được nhấn giữ, thời gian giữ phím, tương tác touch đa điểm… Trong khi đó, xử lý trực tiếp lại hiệu quả với những tương tác làm gián đoạn game (tạm dừng, tiếp tục, thoát game).
* Quản lý xuất hình ảnh bằng hàm GLES2 – lớp CGraphic và lớp CGLES2Driver:
* Quản lý Camera – lớp CCamera:

Do framework hiện tại hỗ trợ cả các phương thức vẽ 2D và 3D, để thuận tiện, ta hiện thực một lớp CCamera chung sau đó kế thừa cho 2 lớp CCamera2D và CCamera3D. Hiện tại, lớp CCamera còn sơ sài, chỉ bao gồm những chức năng cơ bản sau:

* LookAt: xác định điểm đặt camera, điểm nhìn mà camera hướng tới, và hứng nhìn lên của camera. Quá trình này thay đổi giá trị của ViewMatrix của lớp CCamera, ViewMatrix là ma trận sẽ được sử dụng trong CGraphic khi vẽ các đối tượng
* Translate: di chuyển camera. Quá trình này thay đổi giá trị của ViewMatrix của lớp CCamera.
* Rotate: quay góc nhìn của camera. Quá trình này thay đổi giá trị của ViewMatrix của lớp CCamera.
* Frustume: Định nghĩa phương thức chiếu của Camera là chiếu phối cảnh. Quá trình này thay đổi giá trị của ProjectionMatrix của lớp CCamera. Ứng dụng cho CCamera3D
* Othor: Định nghĩa phương thức chiếu của Camera là chiếu song song. Quá trình này thay đổi giá trị của ProjectionMatrix của lớp CCamera. Ứng dụng cho CCamera2D, cũng có thể dùng cho CCamera3D nhưng không tạo được độ trung thực cao như phép chiếu phối cảnh.
* Quản lý IO
  + Quản lý IO là việc đọc các luồng dữ liệu (stream), gồm:
    - File stream
    - Buffer stream (luồng dữ liệu trong bộ nhớ)
  + Về phương thức hoạt động, các luồng chia làm 2 loại:
    - Luồng đọc
    - Luồng ghi
* Quản lý các loại resource:
  + Image
  + Model 3D (with animation) [Phần của Bảo]
  + Sound [Phần của Trình]
  + Button
    - Là lớp mô phỏng một nút nhấn.
    - Thuộc tính chính bao gồm 4 thuộc tính, trong đó có 2 thuộc tính là lớp CImage là 2 hình ảnh mô tả nút khi chưa được nhấn và đã được nhấn, 2 thuộc tính là con trỏ hàm trỏ tới hàm sẽ được gọi khi được nhấn và khi được quét qua (drag).
    - Do hiện thực còn sơ sài, việc bắt tương tác của button trong framework này được thực hiện bằng bắt tương tác trực tiếp với các điều kiện sau:
      * Con trỏ chạm vào trong vùng hiển thị của button và thả ra cũng trong vùng hiện thị đó: thực hiện gọi con trỏ hàm ActiveFunction.
      * Con trỏ chạm vào trong vùng hiển thị của button và thả ra ngoài vùng hiện thị đó: thực hiện gọi con trỏ hàm DropdownFunction.
      * Tương tác được truyển đến CButton thông qua phương thức OnEvent.
  + Particle
  + Font [Phần của Quân]

Camera (Đã trình bày ở trên)

Paritcle(Đã trình bày ở trên)

Các State cơ bản trong game WOT và quá trình hiện thực

Xử lý tương tác người dùng với game WOT

Định kì port lên android và kiểm tra bug

Document:

* Giới thiệu về sản phẩm Game WOT của nhóm:
  + Tổng quát game tower defense
  + Screenshot + gameplay của WOT
* Các kỹ thuật được ứng dụng trong quá trình làm game:
  + Framework cơ bản – cấu trúc chung để quản lý một game
  + Các chức năng mở rộng được thêm vào framework để làm WOT
    - 3D Model (with animation)
    - Sound
    - Particle
    - Button
    - Font
  + Quá trình làm WOT
    - Các task ứng dụng framework để hiện thực:
      * Các state trong game (game gồm bao nhiu state, trong mỗi state có giao diện như thế nào, gồm các thành phần gì, ví dụ như hình ảnh gì, nút gì, chức năng mỗi nút, model gì …)
      * GameObject – CObjectManager (game có các loại object nào, tower, tree, enemy, tương tác - colision giữa các object ra sao …)
    - Các task hiện thực thêm vào:
      * AI của game (mô tả giải thuật A\*, khi nào sử dụng …)
      * Data – Chương trình tạo data cho game (cấu trúc của data, hướng dẫn tạo data từ chương trình …)
      * Resource cho game (Phần nào design, phần nào thu thập về…)
* Kinh nghiệm trong quá trình thực tập:
  + Kinh nghiệm quản lý
  + Kinh nghiệm lập trình