Advance Wars

Specification

Charles Kong
Gary Menezes

Section 1 – Introduction

1.1 Software Overview

This application is a PC clone of the mobile game “Advance Wars”. Advance Wars is a two player strategy game. Each player starts with several buildings. On a given turn, a player can use the income from buildings to buy units. These units are used to capture enemy buildings. A player wins by capturing their opponent’s headquarters, a special building that each player has one of. We built the UI for this game using ClanLib Game SDK, an OpenGL wrapper library that is “a streamlined API” with a “primary focus on games”. We chose this library since it seems to be well documented.

1.2 Motivation and Rationale for Development

Porting this game to the PC will greatly expand the game’s potential user base. It will also increase the game’s actual user base because of this game’s appeal to specific subsets of PC gamers.

1.3 Target Users

The target users for this application are, most obviously, those who have played Advance Wars on a mobile device and desire access to the game on a PC.In terms of gaming paradigms, the target users of this application are the intersection of RTSfans and Indie game fans and PC gamers.This application is also suitable forusers who want ahuman-interactive game with a low CPU overhead.

1.4 Goals

Implement a fully featured clone of Advance Wars. This goal seems attainable, but, to be practical, features are prioritized and those of lesser importance will be cut if necessary.

1.5 Feature / Prioritization / Dev Cost

*A lower number indicates higher priority and Priority >= 1.

Main Menu, Instructions and Game Information:

- Priority: 3
- DevTime: 1 hour

Game Map:

- Priority: 1
- DevTime: 10 hours [includes familiarizing ourselves with ClanLib SDK])

In-Game Menu:

- Priority: 2
- DevTime: 1 hour

Game Object:

- Priority: 1
- DevTime: 1/4 hour

Unit:

- Priority: 1
- DevTime: 3 hour [includes drawing sprites]

Building:

- Priority: 1
- DevTime: 3 hour [includes drawing sprites]

Terrain:

Priority: 2
DevTime: 3 hour [includes drawing sprites]

Player:

Priority: 1
DevTime: 1 hour

1.6 Partners

Charles Kong and Gary Menezes

1.7 Timeline

Sunday, Nov 14th: Meet and work together all day

- Work through ClanLib SDK examples
- Set up complete skeleton (header files)
- Implement a basic game grid and generic units

By the end of Thanksgiving break:

- Have put together all required image files
- Code complete

By due date:

- Debug

Section 2 – Feature design

2.1 Main Menu, Instructions and Game Information

Feature Description

The Main Menu, Instructions and Game Information will be the first item that appears when the user starts the program. It will appear as a text-based menu with options to play the game, read game play instructions, or learn more about the different units, buildings, and terrains that are used in the game. The user will have the option to input via the keyboard which option he would like to pursue. Upon receiving user input, the main menu will either go to the Instructions screen, Game Information screen, or the Game Map.

Functional Specification

The main menu within 100 msec when the user starts the program. Underneath the "Main Menu" title, there will be three options:

1. Start Game
2. Instructions
3. Unit and Terrain Specifications

Option 1 will immediately take the user to game play mode, where the game map with the default terrain and building settings will appear.
Option 2 will take the user to the instructions screen – this will essentially replace the main menu on the user's screen. The user can return to the main menu with a keyboard input of "b" (for "back").
Option 3 will take the user to the Unit and Terrain Specifications screen, which will also replace the main menu. Similarly, the user can go back to the main menu by pressing "b."

Technical Specification

No classes or data structures are needed to implement this functionality. The library that will be needed is <iostream> for user input and file output. The Main Menu will be output to the screen and will continuously wait for user input. There will be two text files, one for the instructions and one for the unit and terrain specifications. Thus, if the user desires to read any of these instructions, the program will direct the user to the appropriate file for read-only usage. While the user is reading the file, the program will continue to wait for input, specifically "b" which signals it to return and display the main menu again.

2.2 Game Map

Feature Description

The Game Map will be a physical 2D rectangular grid shown on the user screen. It will display the various units, buildings, and terrains that are relevant for game play. The map will be represented by a 2D array, and each cell will correspond to an image file (representing a unit, building, or terrain) that will be displayed on the screen. The Game Map will also contain a user interface, responding to user

clicks within the rectangular grid. Finally, the Game Map will keep track of the current status of the game, including the location of units on the map and player turns.

Functional Specification

The Game Map essentially oversees the entire gameplay. It is first initialized once the user leaves the Main Menu and chooses "Start Game." It is based on a 2D grid array. Each cell in the underlying array corresponds to a physical rectangular cell on the screen. These physical cells will display one of the many different saved image files (terrain, building, unit, or a combination). Finally, on top of the physical grid will be a user interface that allows the user to click on various part of the map. Because each physical cell corresponds to an actual cell in the underlying array, the Game Map will be able to discern where the user has clicked, and based on those coordinates, determine what object is currently in that cell.

The Game Map will also keep track of the current status of the game, keeping track of which player is supposed to go. It will interact with the In-Game Menu as well, calling it after a player performs a specific action or clicks on a particular cell on the map.

Technical Specification

As one of the most important and largest features of the game, the Game Map will involve many data structures and libraries. We will be using Open GL to incorporate image files in the screen. There will be one class called Game Map, and within this class will lie many methods that retrieve information on the current status of the game, such as getCell, getUnit, and methods that change the current status of the game, such as invokeAction. The underlying data structure will be a two dimensional array of Game Objects. Game objects include units, buildings, and terrains, which will be described in the following pages.

2.3 In-Game Menu

Feature Description

The In-Game Menu will play a complementary role to the Game Map. It primarily will serve as an additional screen the user will interact with while in the process of building a unit, moving a unit, attacking another unit, capturing a building, or ending his turn. Thus, the menu will appear below the game map only when the program is waiting on some necessary information from the user. It will then relay the information received back to the Game Map so that the Game Map can update the current status of the game.

Functional Specification

The In-Game Menu will allow the user to finish and confirm his move. It will appear as an image based menu that will also receive user input through mouse clicks. In essence, it will appear in the same window as the Game Map, and thus share the same coordinate system that the Game Map uses. However, it will only respond to mouse clicks when:

1. The user clicks on an unoccupied terrain.
2. The user clicks on a base
3. The user has clicked on a terrain to move a particular piece

For the first option, a menu will appear that will allow the player to end his turn. For the second option, a menu will appear that will allow the user to choose which unit to build. For the third option, a menu will appear that will allow the user to either attack a neighboring unit, or to simply move the piece and not attack.

Technical Specification

Similar to the Game Map, the In-Game Menu will rely on Open GL to show images on the user screen. Because there are only 3 possible In-Game Menus, there will only be 3 image files in total. The appropriate file will be called by the Game Map class whenever the corresponding condition has been satisfied. For example, if the current player clicks on an unoccupied terrain, the Game Map will respond to the click, recognize that no unit is currently occupying it, and immediately call the In-Game Menu corresponding to the condition 1. The menu will appear in the same window as the Game Map, so the Game Map can respond to mouse clicks in the region of the menu as well, since they share the same coordinate system.

Immediately following the player's choice from the in-game menu, the Game Map object will perform the action accordingly, and the in-game menu will disappear from the window.

2.4 Game Object

Feature Description

The Game Object will be the most primitive type of piece on the Game Map. As stated above, it is used in the 2D array in the Game Map. The Game object will either represent a Unit, Building, or a Terrain.

Functional Specification

The Game Object serves as an interface for other pieces to derive from it. It will simply keep track of which location on the grid it is currently in.

Technical Specification

The Game object will be one class. It contains no data structures and does not need any additional library.

2.5 Unit

Feature Description

The Unit will be one of the fundamental items of the game. There will be 4 types of units – infantry, mech, artillery, and tank. On a player's turn, the player can move a unit, move a unit and capture a building, move a unit and attack an enemy unit, or not move the unit at all. Each unit will also have information on how much damage it can do to other units, how far it can move across a given terrain, and how much health it has remaining.

Functional Specification

As stated earlier, a Unit is a type of Game Object. There are four types of units – infantry, mech, artillery, and tank. The description of each is as follows:

1. Infantry: $1000 to build. Has the capacity to move 3 spaces. Can capture buildings. Weak against artillery and tanks. Attacker's advantage when attacking other infantry and mechs.
2. Mech: $3000 to build. Capacity to move 2 spaces. Can capture buildings. Strong against artillery, tanks. Attacker's advantage when attacking other infantry and mechs.
3. Artillery: $6000 to build. Capacity to move 5 spaces. Can only attack units that are 2-4 units away. Strong against all units.
4. Tank: $7000 to build. Capacity to move 6 spaces. Strong against all units.

Attacker's advantage refers to the situation in which the attacker gets to deal his damage first to the defending unit first before the defender gets to do damage back.
Each unit will keep track of which player it belongs to, where it is on the map, how much damage it can do to all other types of units, how far it can move across a given terrain, and how much health it has remaining. Each unit starts off with 10 units of health.

Units can only move onto unoccupied terrains or buildings. Units cannot move onto a square in which another unit is already occupying.

Technical Specification

The Unit class will extend the Game Object class and add unit-generic methods such as move and attack. Further, the Infantry class, Mech class, Artillery class, and Tank class will extend the Unit class and then add their own implementations of move and attack. No data structure is needed – only instance variables which keep track of the current status of the particular unit.

2.6 Building

Feature Description

The Building will be another fundamental item of the game. There will be 3 types of buildings – city, base, and headquarters. Each building can either belong to player 1, player 2, or neither. Each player will start with 1 headquarters and 2 bases. Each base will allow a player to build one unit per turn. Every turn, a player will receive $1000 times the total number of buildings he owns. Players can also use infantry to capture neutral buildings or enemy buildings in order to receive more money per turn, to build more units per turn, or to win the game by capturing the enemy headquarters. Each player will start out with 1 headquarters and 2 bases, thus receiving an initial income of $3000.

Functional Specification

As stated earlier, a Building is a type of Game Object. There are three types of buildings – city, base, and headquarters. The description of each is as follows:

1. City: Cannot build units from cities. For each city a player controls, the player will receive $1000 per turn.

2. Base: Each base can build one unit per turn – however, the unit cannot move or attack on the turn that it is built. Each player will also receive $1000 per turn for each base the player owns.
3. Headquarters: Cannot build from headquarters. If the opponent player captures the headquarters, the opponent wins immediately. Headquarters also give each player $1000 per turn. Each player has only one headquarters.

As stated before, only Infantry and Mechs can capture buildings. It takes two turns for those units to capture a particular building.

Each building will keep track of its current status – neutral, in the process of being capture, belongs to player 1, or belongs to player 2.

Technical Specification

The Building class will extend the Game Object class. In addition, each building type – City, Base, Headquarters – will extend the Building class. No additional data structure is needed. Only instance variables will be used to keep track of a Building's current status.

2.7 Terrain

Feature Description

The Terrain will be the third fundamental item of the game. There will be 3 types of terrain – plains, roads, and mountains. Every time a player moves a unit, the unit will travel across one of the above types of terrain. Each type of terrain will contain information on the amount it reduces a unit's mobility. For example, plains are twice as hard to move across as roads, so while a tank can move across 6 pieces of road per turn, it would only be able to move across 3 pieces of plains. Furthermore, each terrain will contain information on the types of units that are allowed on it. For example, while all units can travel on roads, only infantry and mechs can travel through mountains. A maximum of one unit is allowed on each terrain, and units cannot move through terrain occupied by enemy units

Functional Specification

As stated earlier, a Terrain is a type of Game Object. The description of each type of terrain is as follows:

1. Road – Use 1 move. All units can travel across roads.
2. Plains – Use 2 moves for tanks and artillery. Use 1 move for infantry and mech.
3. Mountain – Use 2 moves for infantry and mech. Tanks and artillery cannot travel through mountains.

Technical Specification

The Terrain class will extend the Game Object class. In addition, each terrain type – Road, Plains, Mountain – will extend the Terrain Class. No additional data structure is needed. Only instance variables will be used to keep track of the terrain's type particular attributes.

2.8 Player

Feature Description

The Player will be a class that contains all information relevant to each player's units, buildings, and current status in the game.

Functional Specification

The Player class will serve as a content holder for each player. It will keep track of how much money the player receives per turn, how many units the player owns, if it is the current player's turn, if the current player can make a move (build a unit or move a unit).

Technical Specification

The player class will contain lists of Units and Buildings as well as player data such as income and current balance.

Section 3 – Program Flow

1. Main Menu
    - Start Game
        - GOTO 2
    - Instructions
        - Display game instructions
        - On prompt, GOTO 1
    - Unit and Terrain specs
        - Display
        - On prompt, GOTO 1
2. Game started
3. Player 1's turn
    - Click on Base
        - Options [infantry, mech, artillery, tank]
        - Click on one to build
        - If available funds, start building unit
    - Click on unit
        - Possible moves highlighted
        - Click on space to move to
        - Options [move, move and attack]
4. Player 2's turn
    - Same breakdown as Player 1
    - GOTO 3
5. Headquarters captured/Enemy has no units left
    - Game over
    - Print scores and "Game Over. Player <x> won."
    - GOTO 1

Section 4 – Sample Class Breakdown (public methods and data and select classes)

```
Class GameMap{
        //All unit, terrain, building types
        enumUnit_Type{INFANTRY, MECH, TANK, ARTILLERY};
        enumBuilding_Type{HEADQUARTERS, BASE, CITY};
        . . .
        //Which player's turn it currently is
        Player* turn;
        //Side length of the board
        constIntboard_size = ?;
        //2D array representing the units and buildings on the board
        GameObject*units[board_size][board_size];
        //2D array representing the terrain on the board
        GameObject* terrain[board_size][board_size];
        /*
        All modifier methods, e.g. movePiece(Unit* u, int x, int y), endTurn(), buildUnit(Base* b,
        Unit_Type u)
        */
}
Class Menu{
        //Display the main menu at a fixed position
        voiddisplayMainMenu();
        //Display the move menu at the given x,y coordinate
        voiddisplayMoveMenu(int x, int y);
        //Display the (build) unit menu at a fixed position
        voiddisplayUnitMenu();
}

Class Player{
        vector<Building*> buildings;
        vector<Unit*> units;
        int income;
        int balance;
        . . .
        voidstartTurn();
        . . .
}
//Has no outside references other than its Player
Class GameObject{
        Int x, y;
        Image* i;
        Player* owner;
        intmoves_left;
        . . .
        void move(int x, int y);
        void attack(GameObject* o);
        . . .
}
```