Jong Seong Lee
CS506 Proj: Parallel Game Development
Project report final
Summer 2017

Project Introduction

One of the projects I have been worked on in Computer Science courses at PSU is to develop the simple web game called Baby Trader using the HTML5 game framework called Phaser.js. It was for a term project of Open Source Development class, and the main purpose of this game is to sell babies to customers. Customers come into a player's store and would ask for babies with specific talents such as being smart, handsome, or potential jobs in the future. A player has to pick a baby from his or her selection and propose it to a customer. If all conditions match, he or she will accept and buy the baby; otherwise a player should recommend another one. If there is no baby matching, a player can cheat and change attributes of babies using the special button on screen.

The idea of this game is interesting but the original game is too simple and almost hard to call it a game because a player just needs to deal with only one customer at a time and browse babies to make sales. The purpose of this project, Parallel Game Development is to upgrade the old game using multithreading technology. In the new game, everything is new except some old graphic elements. The target platform of the new game is Android, and the game engine framework, LibGDX is used.

What is LibGDX?

Since Android mobiles are introduced, a lot of games have been developed for the platform and there are many frameworks out there for developers. LibGDX is one of the best 2D Android game framework, and it also supports many other platforms such as Windows, Linux, Mac, IPhone, HTML5, and Blackberry. LibGDX is an open source game development framework that works with Java along with some C and C++ components.

The founder of LibGDX, Mario Zechner, who is also the author of Beginning Android Games, is an expert of Android programming and many of his skills and techniques are implemented in LibGDX.

The biggest advantage of LibGDX is that being cross platform. Developers need to change only a small portion of code to add another platform in their game, and programs would work exactly the same no matter which platform they run on. So when someone develops an Android game using LibGDX, he or she does not need to have an Android emulator program or an Android phone itself because a developer can debug using only desktop. This feature offers higher productivity and faster development. A developer can use powerful debugging functionalities of JVM too because of Java.

Game contents and mechanics

The game consists of a few different screens, and each provides specific features.

- Logoscreen
  Shows the logo image for three seconds before landing on the main menu.

- InitScreen
  This is the main menu that a player may choose several options by pressing or touching the buttons. It has three buttons that switch the screen to the slider screen, the credit screen, or the tutorial screen as well as the button that exits the game.

- CreditScreen
  Simply displays credit information regarding the developer, the game engine, and the music resource. Touching or clicking on the url texts opens each website on a browser. Also it contains the button that switches the screen to the main menu.

- TutorialScreen
  A few pages of tutorials for beginner players. Each tutorial page contains the description with the screenshot that explain how to play. On the bottom right, there are two buttons for switching the screen between the pages, so a player can view multiple pages of tutorials. On the bottom left, there is the button that switches the screen to the main menu so a player can exit the tutorial.

- SliderScreen
  There are two slider screens and the purpose of having them is to set up attribute values that are used for the game play. An attribute describes a baby's talent, and a set of attributes are categorized into positive and negative groups. That is why there are two different slider screens; one for positive and the other for negative. A player may use a slider to control selling and buying values for corresponding attribute, and this affects his or her game play. Default values are populated randomly every time a player enters into the slider screen, but he or she does not have to modify default values. When a player finishes setting, this will take him or her to the ready screen.

- ReadyScreen
  There is no feature except displaying the "ready" text for three seconds and switches the screen to the game screen. The purpose of this screen is to avoid surprise for a player by giving some delay before it begins the game.

- GameScreen
  When Ready Screen is disappeared after three seconds, a player may begin playing the Baby Trader game. This is the most complicated screen compared to others as it involves many elements and features.

The hud on the right contains the face of the Baby Trader, timer, the current balance, and three buttons. The face of the Baby Trader has no specific feature but it is a visual notification of game events.

When a baby is sold or purchased, the game screen will catch the event and change the face of the Baby Trader from normal face to smiling face for a short period of time. Visual notifications are important for this game because trades are automated by multiple threads.

Timer simply counts down from five minutes to zero, and the game is over when it hits zero. The current balance displays the amount of money that a player has, and the game will be over when money is less than or equal to zero. A player's money value will change depending on events such as selling and buying babies.

The hud comes with the promotion, research and menu buttons. The promotion and research buttons allow a player to upgrade levels of the promotion and research abilities of the game by spending some amount of money. These buttons will open the upgrade popup and display amount of money needed for upgrade as well as text description, current level and next level. If a player has enough money, he or she can press the confirm button to upgrade his or her desired ability. Otherwise, it will display warning message that a player does not have enough money. A player may also close the popup window by pressing the cancel button. The menu button works similar to the other buttons, but its main purpose is to pause the game and exit if a player wants to finish his or her game. All these popup windows will pause the game until they close.

In the middle area of the screen, a player may browse babies in stock. The information of a baby displayed here are sprite, name, age and a list of talents. A player may browse babies by pressing arrow buttons, and number of babies available is shown between the arrow buttons so he or she can see how many babies are there for sale.

On the left, there are two sections where one is displaying customers buying babies from a player, and the other is customers selling babies to a player. Both sections look almost identical; they display a customer's sprite, name, age, and a list of talents they would like to have on their baby or ones belong to babies they would like to sell to a player. Also each section has the counter of customers waiting in line as well as the upgrade button that opens the popup window for ability upgrade. The upgrade popup windows work the same as ones for the promotion and research upgrade; however a player can see his or her level of selling and buying ability without opening the popup window.

A customer sprite for each selling and buying section gets replaced with either "sold!" or "purchased!" text when each sale event is triggered to notify a player because, without a notification like this, a player would have hard time to see what is happening in the game.

- GameOverScreen
  The game over screen will appear when the timer or the balance of game money becomes less than or equal to zero. This screen displays the high score and a player's score, and each score has a player's amount of money, number of babies sold and purchased, and number of customers visited when the game was over.

  This screen will check if a player's score is greater than the high score saved in the game. If so, the old high score will be replaced by new one; otherwise, it won't save the new score. Comparison depends on the four different records:

  - Game money balance when game was over
  - Number of babies sold
  - Number of babies purchased
  - Number of customers visited

  The high score is calculated by the following formula:

  High score = game money balance + (number of babies sold * number of babies purchased) + number of customers visited

  There are two buttons that a player may press to go back to the main screen or retry the game. If a player choose to retry, it will switch the screen to the slider screen.

Why this game needs parallel execution?
The upgraded Baby Trader contains multiple threads to make the game more interesting and complicated. Baby Trader has several different components that are represented by threads:

- SalesTeam
- PurchaseTeam
- ResearchTeam
- PromotionTeam

Each component is executed by a single or multiple threads. When a customer comes into a player's store, he or she visits the sales team or the purchase team to either buy a baby or sell one. The sales team sells babies from the baby stock and the purchase team buys babies from customers and add ones in stock. As the sales team and the purchase team may add or remove babies, the baby stock is dynamically changed by the sales and purchase teams.

For the sales team and the purchase team, a player can improve their abilities by upgrade. Each upgrade will add another thread of the sales team or the purchase team. Having more than just one of them is very similar concept to having multiple sales representatives in a store. That way, number of customers waiting in line would decrease and productivity will go up. A player can make money faster because he or she can deal with multiple customers at the same time.

Each sales team and purchase team has a feature to notify a player when trade is successful. This is important because all trades are automatically done in the game, and a player would not know what happens without notifications. When there is a sale or a purchase, it will set the flag for a short duration of time. Then the screen rendering class will catch the flag and display the special graphics to visually notify a player. When the flag is disabled after some time, the screen rendering class will disable the special graphics.

In the middle of the routine, the research team takes an action to make babies better products by replacing negative talents of babies with positive ones. For example, the research team can replace “ugly” attribute into “nice job” attribute of a baby that a player has in his or her stock. This will increase the chance of sale and the value of the baby because a customer wants to buy babies with positive attributes. The research team will change negative attributes of one baby in the store stock every once in awhile. It is also possible to upgrade the research team as well; doing so would make the research team replace negative talents of babies with positive ones more often. That way, a player can expect higher score.

Without the promotion team, other teams cannot perform anything because there will be no customers. The promotion team brings a customer to a player’s store every a few seconds, and it happens individually on its own thread. It creates a buying or selling customer and add him or her in the waiting queue. Upgrading the promotion team makes it brings customers more often. However, just upgrading the promotion team will cause having too many customers waiting in the queue.

Java multithreading
Java offers the class, “Thread” for multithreaded programming. Thread class represents a thread in execution and offers a set of methods for control. Programmers who have multithreading experiences with other languages would find similar method names.

The constructor of Thread object takes a runnable object to be initialized. A runnable object inherits a Runnable interface to implement run() method. This method will be triggered when Thread’s start() method is invoked. In run() method, it is common to find looping and waiting operations.

To terminate Java Thread, interrupt() and join() methods can be used. The interrupt() method is used to stop a running thread, and it could throw an InterruptedException depending on a situation. For example, if a thread is interrupted in the middle of waiting for sleep(), the thread will stop its execution and throw an interrupted exception. A programmer can use join() method to destroy a running thread and merge multiple flows of execution into one.

For communications between threads, it is inevitable to use static data because nonstatic variables would be thread local. And data being static can cause race conditions because if a thread’s data access is not atomic, it is hard to know whether data is valid, and a program’s behavior would be unreliable.

In Java, it is fairly straight forward to take care of such problems by using synchronized methods or blocks. Any operation within synchronized methods or blocks is treated as atomic and other threads waits for the operation to be over.

Because every operations are atomic and other threads should wait for synchronized methods, it would be a better idea to use synchronized blocks if target operations are long and contains both data sensitive and insensitive behaviors.

There are three classes in this game that use synchronized blocks and methods.

- The share data is a static class, and it contains the set of data that is shared by many classes. It could be shared by either runnable or non-runnable classes, and it has to be prepared for race conditions.

  The constructor of the share data, getters and setters are synchronized methods since they do not have complicated operations. There are a couple of non synchronized methods that contain synchronized blocks instead, where contains a bit more tasks involved. Declaring returning value and return statements are outside of synchronized blocks since they are not directly influenced by shared static data.

- The game screen is the class that draws game screen, and it also has one synchronized block in its rendering function. Mostly this class would not need any synchronized operation, but it is needed when browsing babies. Game screen has that feature that a user may browse baby stock in real time by clicking arrow buttons. Which means that the game screen will keep the index of the baby list that is manipulated by multiple threads. This may cause the index out of bound exception because the size of the baby list can be changed at anytime.

  For example, the game screen pointing at the max index of the baby list could fail its operation if the size of the baby stack gets changed between two operations, getting baby index size and getting a baby by a index value. So game screen has synchronized block that wraps the portions and make them atomic.

- The research team replaces babies’ negative talents with positive ones every once in awhile. Which means the operation of this class is data sensitive. It searches for a baby with negative attributes, then performs its action. But the baby that the research team found may be sold in the middle of operation if it is not protected with a synchronized block. Since this class is runnable, it contains a waiting operation to give some delay between a sequence of operations, but the synchronized block does not wrap waiting operation because it will cause massive delay.

LibGDX and multithreading
In LibGDX, rendering is taken cared by a separate thread. Any screen rendering class should implement “Screen” interface provided by LibGDX, and it has an interface method called “render”. In render function, a screen rendering class continuously draws background color, graphics and texts.

To execute some logics before render function, a developer should use “Gdx.app.postRunnable” function that takes a runnable object as a parameter. In this way, a run method of a runnable object will take the first turn before screen rendering happens.

There are a couple places where Gdx.app.postRunnable method is used for this project:

- PromotionTeam: The run function of the runnable object that is passed to the postRunnable function simply adds either selling or buying customer and increase count of customers visited.

- ResearchTeam: Inside the run function of the runnable object passed to the Gdx.app.postRunnable function, it has the synchronized block that manipulates baby talents. First it searches a baby with negative talents, then it switches any negative talent with positive one. This happens before render function executes, so it can finish its job before graphics are updated.

Screen interface also has some important methods that might affect multithreaded programming:

- pause(): This method is called when a game screen is paused.
- resume(): This method is called when a game screen is resumed.
- hide(): This method is called when a game screen gets switched from this one to another one.
- show(): This method is called when a game screen gets switched from another one to this one.
- dispose(): This method is called when a game screen is no longer used and gets destroyed. Similar to destructor and often used to invoke dispose methods of source objects such as Texture.

The most important methods among them are the hide and the show functions because they get invoked when a game screen gets switched from one to another. Unless a developer wants to keep old data, the show method should be treated as an initialize method. The hide method does not have a specific role other than being a semi destructor, but it is important when a developer uses runnable objects along with screen classes.

Any thread used in screen class should be destroyed in a hide method using Java's interrupt and join methods. If these operations are not performed, a screen class creates a new set of threads that performs the same actions when initialized. This will lead the game to perform the same actions multiple times when a player lands on the same screen that he or she visited before, and this can cause an unexpected behavior. For example, if a developer uses a thread to count down a timer, it will count down two every second instead of one if a player visits the same page twice.

For this project, the game screen class that implements Screen interface of LibGDX contains a function that kills threads that this class uses in its hide method. This way, it can avoid accidental actions caused by a set of duplicate threads.

Bugs caused by multithreaded programming

- Trade notification was out of sync

  This game notifies a player when trade is successful because it is hard to see what is happening since almost every action is automated. So it is important to visually show when babies are sold or purchased.

  First it was handled by the class called the event tracker and the purpose of this class was to play a sound and set a flag so that the game screen class can display special graphics for a short period of time.

  However, the number of notifications and the count of actual trade did not match. The event tracker waits for a half second when it catches successful trade and some other trade bypasses the event tracker while it waits. And the event tracker was almost redundant class because its job can be handled by the sales team and the purchase team instead. So the event tracker was removed on the later stage of development.

  The logics of the event tracker moved to each class, the sales team and the purchase team. Then, both classes contained Gdx.app.postRunnable methods, and putting the logics inside the method didn't really work because of the sleep() method being a part of the logics. The Gdx.app.postRunnable method runs before rendering and it caused a massive delay when it waits.

  The event tracking logics should be out of the Gdx.app.postRunnable methods, so next try was to move only the event tracking logics out of the method. But that also didn't work because trade happens in the Gdx.app.postRunnable method and there is a delay before the event tracking logic happens. So in timing wise, it didn't really solve the problem.

By looking at the Gdx.app.postRunnable methods of the sales team and the purchase team, their jobs didn't really have to happen before rendering, so I removed the Gdx.app.postRunnable of each class. Before then, my understanding was off, and I thought the Gdx.app.postRunnable is a method that a multithreaded logic should call to push data, regardless of precedence of execution. It was okay to use it but not required in the sales team and the purchase team. Now notification system works well and synchronized.

- A duplicate set of threads

  Before understanding how Screen interface of LibGDX works, my thought was that any screen class and its contents should be destroyed when screen switches to another. So my code didn't really destroy any thread that runs along with the game screen class.

  This caused a problem because when I switch back to the same screen that I visited before, previously generated threads remain, and the same actions happened multiple times. If the same screen is visited three times, the same action happened three times, and it was a sign that threads were not appropriately destroyed.

  After researching and understanding on how Screen interface works, thread killing logics were placed inside the hide method, which it gets called whenever screen gets changed to another. However it didn't solve the problem.

  The issue was how hide method was trying to kill threads. My initial understanding of killing threads with Java was only to call the interrupt() method of Thread class, which should stop its execution. However, the join() method was required to completely wipe out and merge multiple flow of executions, so separate threads do not remain when the same screen gets displayed next time.

  So inside the hide method of the game screen, after calling interrupt() for each thread instance, join() method is called for each one of them as well. And the problem is disappeared and there is no more garbage threads remain when the game screen switches.

  The interrupt() method of Thread class causes the interrupted exception for those threads wait using the sleep() method in the middle of execution because they are bothered by interrupt() methods while waiting. However, this is not an error and expected to be happend. This game may print the message of exception in log, but it should be ignored.

- The game did not completely paused when separate threads are running

This game uses multiple threads to update graphic elements of the game screen, and updating graphics should stop immediately when the game is paused. However, it didn’t stop right away when the game is paused by a player. It stops, but there was small delay before it happens.

Each runnable class uses the sleep() method in the middle of iterative operations, so it does not run too many times. Also it checks if game is paused too, so it does not execute its job when the game is paused. Before fixing this bug, the sleep() was located after checking pause status. This cause the issue because after the sleep() is over, whether game is paused or not, it performs its operation.

This bug is fixed after moving the sleep() before checking pause status, so it never runs when game is paused after sleep() is over.

Other components of game

- Configuration

  This class contains data that is not thread sensitive, although they may be used by threads. It contains number of babies sold and purchased, number of customers visited, current level of abilities, and some flags for notification system.

- Attribute

  The talents of babies are defined here. This is a enumeration instead of being a class, but it has a few functions that help settings. The most important one is the reroll method that it randomly populates prices for selling and buying values for each attribute every time when game starts.

- Person

  A parent abstract class for the baby and the customer classes. Each of them contains metadata such as name, age, a list of talents and a graphic sprite for displaying on the game screen.

- SaveData

  This game can save data locally by using this class. The main purpose is to save the high score as well as amount of money, number of babies sold and purchased, and number of customers visited.

- SharedData

This class contains data that is thread sensitive. All data access operations are protected by synchronized blocks or synchronized methods in this class. It manipulates amount of money, babies for sale, and customers visiting store.

- Timer

  The simple class that works as a timer.

- Popup

  Popup that is used for the game screen class. It is a collection of graphic elements, buttons and texts.

- CommonUtilities

  The collection of helpful methods like random number generators.

Conclusion
Multithreading is a subject that many programmers are confused about and it is often considered too much for programming. It is not too common to find multithreading in ordinary development environments. However it is true that multithreading is involved in more places now and there are anticipations of higher demand of multithreaded programming skills in the future.

There are some multi threading courses in school but I believe I had a chance to have a good amount of hands on experience of multithreaded programming with a subject that I enjoy and practical language with this project. Java is commonly used in many development environments and it was a nice opportunity to learn how Java multithreading works. This will make me less hesitant to try out and dive in multithreading development and study for further multithreading research subjects.

Reference

- LibGDX (https://libgdx.badlogicgames.com/)
- LibGDX wikipedia(https://en.wikipedia.org/wiki/LibGDX)
- Starting LibGDX (http://edoli.tistory.com/139)