Assessment 4 Extension Report

The extension report contains information detailing the challenges faced by Team EEP in extending and modifying Team ADB's TaxE game for assessment 4 [10], as well as the software engineering solutions and approaches adopted by Team EEP.

Software Engineering Solutions and Approaches Adopted

The Main Approach

Similar to the last assessment completed, the team has used KISS approach (keep it sweet and simple / keep it simple stupid) to meet the clientele's requested requirements. The clientele have made specific requirements for this development stage which must be completed over a period of nine and a half weeks (23 Feb - Apr 29th). The work was split into 10 scrum sprints that lasted for one week each (except the last, which lasted half a week). As the deadline was fairly short and team members had other commitments (open assessments, exam revision, societies, other work, etc), the team made it a high priority to only implement what was required and implement these to the highest possible quality with explicit details discussing what, how and why it was completed. There are two main features that needed to be included which were split into Git branches respectively: replay mode and track modification. This stopped confusion occurring as to the progress of a feature and ensured one feature was fully functioning before being integrated into the master branch for the final product. When both features were developed, a completed TaxE game was presented to the clientele based on team ADB's requirements specification (assessment 1 and 3), FVS's requirements specification (assessment 1) and previous meetings with the clientele for this development stage to ensure we exactly met their requirements. The added features are fully documented and coded to a professional and highly readable standard. They are presented in a manner for future developers to extend and maintain the game for the clientele with as little hassle as possible.
Currently, team EEP has six team members. The two main requirements (replay system and track modification) and the one preferable requirement (cross-platform mobile version of the game) were assigned to distinct team members. The team member was responsible for completing their set task and using the scrum sprinting technique, will report back on the progress of the feature when a sprint ends. This will enable time and resources to be distributed appropriately based on the progress made in the features where more will be allocated if the feature being implemented is falling behind schedule. Two of the remaining team members, including the group leader, concentrated their efforts into writing the documentation. Any other features created by developers not allocated a specific requirement will contribute and explain their own modifications to aid these members (store information in a change log) and review each others work to keep the documentation at a professional industrial standard. This will help the readability and maintainability of the system for the future by creating an understandable and clear piece of documentation about the modifications. The final team member worked on the team's website, as we intend for this to be an important part of the group's presentation from a marketing and business perspective when showing the clientele and to entice the audience to download the game because the website will most likely be their first destination to learn about the game and download it.
Ideally, the development team will be closely following the requirements specification of team ADB and team FVS as they were the requirements stated by the clientele. The team will clearly identify the requirements needing to be achieved at this stage beforehand, however, some information may be unknown and not documental until it has been discovered or fully implemented. The team will hence follow Parnas and Clements paper [9] in the sense that if some information in the documentation is unknown at the time, then it will be implemented at a later date as, according to their findings, it results in a high quality product still.
Overall, our team needed to create an extended TaxE system that met the new requirements stated while being efficient (processing and space), maintainable (easier for developers to fix problems), readable (easier for developers to understand) and extendable (easier for developers to extend the game in future development stages). The game was also coded to be secure to stop anyone playing the game from cheating (e.g. changing their train speeds), which would have otherwise created a reduced gameplay experience for the user not cheating. Decreasing any of these qualities would have reduced the quality of the final software and, as the clientele wanted to attract as many users as possible, the system needed to be of the highest quality. These qualities apply to both the underlying code and GUI. The testing approach is noted in the test plan [11].

Change Management

Change management in a software engineering context attempts to control control, manage and support changes to software [13]. This is done by gathering requirements, determining whether they are attainable, planning how to go about implementing the requirements, implement them and then evaluate the changes. Its main aims are to support the processing and traceability of changes to an interconnected set of factors [14]. The process can hence be handled in an agile manner as it methodology is based around volatileness.

To comply with change management, our team initially gathered requirements by reading the newly set requirement changes given by the clientele on the SEPR webpage [15] and setting up a meeting with the clientele to find out if any additional features were requested. Based on these requirements, a priority was given to determine whether it was realistically achievable with the time and resources available. The highest priority was given to the requirements on the SEPR webpage, as these were mandatory, while the other requirements were given priority based on the clientele's and team's opinion (e.g. based on what the clientele truly wanted, the programming skill of the developers, time available, etc). This may otherwise negatively impact the clientele more by having a piece of software not catered to them specifically. In relation to this, the highest priority requirements were planned first and once these features were implemented, the next feature in the priority list would be planned and so on. These were planned by distributing team members to said tasks based on their ability (e.g. a team member skilled in programming would do a programming based task while a team member skilled at documentation would perform a documenting task) and the amount of team members necessary to allow an even distribution of resources and time to ensure the final deadline is met with the bare minimum at least. A gantt chart [21] was planned to help track the remaining time on the project and when features should be finished in coordination with the scrum sprints. Not using the plan would otherwise have negatively impacted the software's quality by having a higher risk of delays, which in turn would mean quickening the pace of finishing pieces of work before the final deadline.

Agile Development (Scrum)

The software development team decided to continue using the agile development approach previously adopted in the past development stages (assessments 1, 2 and 3) due to its current success within the group and how it fitted into the current project style and structure. Agile development is based around the concept of requirements being volatile with a focus on the clientele and adapting to their needs over the entire development through constant communication, such as meetings. Due to the focus of the project being on the clientele's requirements, an agile development approach will ensure changes made to the software happen efficiently, using the least amount of resources and time as possible. As it had been previously used too, no time was needed to train team members into how the approach works, thus work could begin instantaneously.

As an agile development approach, the team decided to continue using the agile development method of scrum. Scrum involved short iteration cycles called "sprints" in which goals are set to be achieved after each sprint by specific team members. Each sprint tended to last around a week and all updates from team members were directed to the scrum master (Project Leader) at the end of the sprint. The scrum also acted as a tracker to check each team member's progress throughout each sprint and the project as a whole. Scrum was chosen because it was a familiar technique for the team, having been used in the previous development stage in which effective results were produced that resulted in a high quality system. By using a familiar technique, team members already understand what was going to happen when objectives were distributed and how long they had to complete the tasks. As a result, resources and time were issued elsewhere due to saving time from learning a new software engineering approach. In addition, each team members progress could be monitored so that, in the event that someone falls behind, more resources (team members) can be assigned to the area required. This stopped the project being delayed and guaranteed the project finished by the deadline, otherwise the team would not have met the clientele's requirements which, in turn, would have decreased the quality of the system. In addition, there was no point in altering a software engineering approach if all team members were happy in continuing to use it and it would have confused the team members if a sudden change occurred.

The test-driven development method was used and is explained further in the test plan [11].

Refactoring

Refactoring is a controlled technique for improving the design of an existing code base [5]. Small improvements could be modifying variable names or adding comments to increase the readability of the program while larger improvements might include altering how a search is conducted (e.g. from linear to binary) to process the search faster. By implementing small iterative improvements, the system can slowly be modified and enhanced, accumulating to become a more efficient and professional system for the clientele when the version is released. This can be achieved by improving the readability, maintainability, security, space and time efficiency of the system. Primarily, refactoring will be conducted by each developer on their piece of coding, however, the lead programmers will check and refactor all coding that is deemed to be the highest of priority. The priority order was determined by having any initially requested clientele requirements at the highest priority (i.e. the replay system and track modification) and any further requirements placed at a lower priority and ordered based on the team's resources, time and the clientele's recommendations (what feature they think is more important to implement). Refactoring phases were considered after implementing or modifying a feature (and every two days) and were reviewed by the lead programmers as to whether it was functioning or not before merging it. One final refactoring phase took place two days prior to the deadline (Monday 27th April) to verify the system is fully functioning and as efficient as possible for the clientele's final product.

Unfortunately, our team did not have enough time to refactor all of Team ADB's code due to the code size of the system as we prioritised implementing the main clientele requirements. The system will not be as efficient as it could be, reducing the quality of the coding system, but still fitting the needs of the clientele, which would improve the clientele's satisfaction.

Tools Used

This section details the tools used within this development stage. A list of the test tools used and the IDE chosen can be found in the test plan [11].

Gradle

Gradle is an open source build automation system. Gradle can automate the building, testing, publishing, deployment and more of software packages or other types of projects such as generated static websites, generated documentation or indeed anything else [1]. Team ADB used Gradle in the previous implementation stage to build their project, so our team will need to continue using it to add and maintain features for the system in an effective way. The downside is that, because our team has not used Gradle very much, time will need to be allocated to training into understanding the software, which can be done by researching into it ourselves or asking Team ADB how and why they used Gradle in their system. The latter would be the simplest option and save us more time and, seeing as the development stage only lasts two month, any more time towards other aspects of the system would be valuable. Luckily, it should not take as long to learn/relearn as the team has worked on Gradle in the previous assessment.

LibGDX

LibGDX is a game-development application framework written in the Java programming language [2]. It is primarily used to create desktop and mobile platform games, hence it is cross platform. By being cross platform, the TaxE game can target the largest possible audience range without limiting what operating systems can be used. Team ADB and team FVS both used libGDX to create their visuals in the previous implementation stage and so the team will need to continue using it to maintain and add features to the system, particularly the GUI aspect. As no team member has used libGDX before, a lot of time and resources will be required to understand and train the team into learning it which could cause problems in delaying the project if too much time is spent on this. The alternative is to use Java Swing as we are familiar with that framework, however, we would need to change the underlying architecture of the system. This would take far too much time and resource to implement based on our short deadline, hence the team decided to continue libGDX. The benefit is that libGDX is open source and popular, meaning there is a lot of support on hand if there are any questions that need answering (as well as team ADB being available), which will aid us in implementing the system with a framework that the team is still fairly new to. However, the team should not require support as often or take as long learning new features of libGDX as the team has previously worked with libGDX on the previous assessment.

Previous Documentation and Implementation/Availability of Team ADB

To aid our extension of Team ADB's TaxE game, the development team had requested the documentation to be provided to our team, including all code for the system and the website. Team ADB were also available by email (University of York email) and face-to-face communication if necessary. Both of these provided useful support in understanding the current version of the game, which in turn helped us to extend the game and modify it to fit the requirements of the clientele at this development stage. It did, however, take a considerable amount of time to study the documentation and code initially because it was a large code base, hence appropriate time was allocated (around two days). As a result of reading the documentation, the system was made to the highest possible quality and also saved us time and resources by not needing to create features that were meant to be previously implemented or by using their resources, such as the current UI, as existing templates for our system.

Git/GitHub

Github is a web-based repository hosting service for the source code management and revision control of Git. It provides a visual representation through an interface for developers to collaborate on projects and share code in a more efficient manner. The code and its versions are much simpler to maintain between all the developer's systems by working on separate branches that will not intervene with one another when working on each of the three features to implement. These can be merged by a dedicated merger (an experienced programmer and GitHub user such as one of the lead programmers) when fully functional to a master branch to ensure that a working solution is always available in the event that an error occurs and the code needs to be reverted back to a working implementation to resolve it. A dedicated merger will also ensure that no branch overwrites another branch someone is working on or that any issues are present when merging. They will be aided by the continuous integration service Travis-CI [6] that is embedded into our repository to test if the integration between branches can occur. Details about Travis-CI can be found in the test plan (section 7.4) [11].

In addition, the repository on GitHub is open source and where team ADB's website and TaxE code are present, hence we have instant access to their coding that we can use to fork our own repository without having to specifically ask and request it (only for the website so we can modify it with additional information). Every developer will have access to GitHub to enable the individual to have access to the code whenever they want it, while a dedicated merger will be used to manage and approve all branches and extensions to the code. Before they are approved, the dedicated merger will see if the code correctly works with evidence of their testing and then check whether there will be any issues in the merge. These will be pointed to the developer who made the merge request to fix with the support of the dedicated merger. Up to three days will be given to fix the issues where additional resources and time will be redistributed to here depending on the progress of other tasks.

Challenges faced

The development team have encountered several challenges when implementing code to complete the requirements of the clientele. Changes have been split into three change categories: perfective (architecture/algorithm improvements), corrective (fixing previous code/errors in the version) and additive (adding in new features). Changes in the code are denoted by comments being denoted by "//New methods by Team EEP" or by a developer and time stamped beyond 23/02/2015. A list of all the modifications are detailed in the change log [Appendix P].

Perfective Changes

Lack of MVC Structure

Initially, to understand the system before attempting to make any changes, our team studied the extension report of ADB's system created by our team, FVS's architecture report [17] and team ADB's documentation. Unfortunately, team ADB did not have a class model diagram to follow, which became awkward when the change management occurred as it took longer than normal to understand the system's structure with no supporting diagram to constantly reference/relate to when building the system. In Team FVS's architecture report [Appendix O] [17] (the originally extended TaxE game for assessment 2 before ADB in assessment 3), it was stated that their chosen architecture style followed a MVC/DRY hybrid (Model View Controller/ Don't Repeat Yourself) [Appendix O] approach to reduce code duplication and complexity. In hindsight, team FVS found it was useful in the short-term, however, it has become problematic, not impossible, to extend the features for our current implementation stage. As ADB stated within their extension report [18], "it was difficult to identify where components of the game should be located in the code and this was due to the incomplete separation of view from controller in their MVC format". In turn, the architecture caused issues with the understanding and readability of the code with regards to how the implementation works. where certain functions are located and where to implement our new functions. This increased the amount of time spent coding each feature and increased the pressure on the team, which may have resulted in a lower quality product. Luckily, there was still time and resources available to allocate to reduce the impact of this issue, however, to prevent it occurring in the future, a complete restructuring of the architecture would be in order. Based on the time frame for the assessment, this was not possible as it would take up too much time in reprogramming and testing it as it would surely create more bugs to fix within the program. Overall, the team managed to code around the architecture problems that persisted and attempted to resolve the issue during their coding time. For example, the Actors are now out of the Models and are referenced from the *ActorsManager* class. The code base was also particular large which meant the probability of errors and bugs being contained in the code was much larger, therefore the development team could not spend as much time adding new features to the game because priority was given to the replay system and track modification features first.

Corrective Changes

Resolution

When initiating the game from one of our laptop screens (averaging 1366 x 768 in resolution), the GUI was loaded into a smaller fixed resolution (1022 x 678) [Appendix M, 1]. This made the GUI playable on all screen resolutions, however, the game did not feel as immersive as it could have been or have a dynamic environment for the user to adjust as they wish by having the fixed resolution. The user may therefore feel constrained to play the game in a way that the developers wished or felt easiest rather than giving the user the freedom to alter the screen resolution (like in most modern day games), hence it may deter users from playing the game which is not what the clientele would want. Also, due to the constrained window size the game, the resolution could not support smaller screen devices and could not be further developed for use with mobile platform, hence changes had to be made based on our developer perspective. As a result, the team decided to implement changes to support the ability to resize the game's GUI by adding a StretchViewport to the Stage object of GameScreen used in the game as this was a method that the developer knew how to implement, tutorials were available as support if required and no other way seemed to fit the task appropriately based on our research. The viewports enable the game screen and UI elements to be resized so they fit different screen sizes, while preserving the ratio of the UI elements to the ratio of the

resolution they were originally developed to be for (1680 x 1050). However, with this implementation, whenever the game was run on a different resolution to 1022 x 678, all of the images and text would be blurred, which would lower the graphics quality and hence the overall quality of the system. It also prevented the user from having the freedom to choose or stretch the window to their own resolution, making a restrictive gameplay experience.
In addition, a CustomTexture class was added to define a Texture object and apply a linear filter to it upon creation. The reason being is that whenever the screen is adjusted in size by the user, all of the images used for the UI would not suffer from quality loss as they would be filtered first, keeping the system consistent and professional. A linear filter was also applied to all the fonts used in the game for the same reason. The changes were implemented as they would enable users of all screen resolutions to play the game, which would increase the audience range and gameplay experience, and the problem was shown to the clientele, which was highly recommended to fix.
The game also had several UI elements(e.g. connections) which were drawn using batch (their draw methods were overridden). The above techniques wouldn't apply to them as they were drawn in a static coordinate system and whenever the game screen was resized, they would remain the same size. In order to make those elements resizable, the projection matrix of the GameScreen stage was passed to the batch responsible for drawing them. With this fix the game achieved full resizability to provide the user with the freedom to resize the game's screen and hence allow us to use this for creating a mobile version of the game.
The change can be seen in the testing evidence as well as in the appendix [Appendix M], showing that the change was successfully implemented with no errors.

Train Graphics

A minor change occurred in the Assets folder whereby new train images were added to replace the existing train images in a PNG format. These new trains are displayed in place of the old trains [Appendix M, 2]. The reason is that the original train images were classified by the team as looking like a "five year old had drawn them" (in which the clientele agreed with), hence it was associated with being of a poor image quality which may deter users playing the game based on low quality aesthetics. In response, higher quality images were found that were accessible and free to use to avoid potential lawsuits against the clientele. The images were in a PNG format, allowing flexibility with the image through lossless compression (quality not lost each time images are saved or reopened) and supporting more colours (up to 48-bit true colour [16]) to maintain a high standard of quality for the clientele's product and to immerse the user more into the game. No new tests were required.

Inheritance Issues

Whenever a user modified a track's connection by adding or removing it from the map and game, a bug caused the game to crash. One of the team's developers debugged the code and located an issue that was caused due to a lack of inheritance in the *StationActor* and *CollisionStationActor* (the junction actor) classes. *CollisionStationActor* was not extending *StationActor* and there was code duplication, having two identical methods which draw each of them. The team decided to refactor the classes to use inheritance in order not to introduce more code duplication with the connection modification methods. Although this inheritance issue might not have been a big problem in previous assessments, it was essential for completing the track modification requirements. This issue caused a lot of confusion for our developers, as it turned out to be really hard to find the problem and fixing it helped to solve a lot of similar problems that could have occurred for future developers, helping our system become highly maintainable, reliable and extendable with ease.
To correct the issue, a new constructor (Instantiation method) was added to the *StationActor* class (line 43) to enable the *CollisionStationActor* to extend *StationActor* appropriately with a different texture image to represent a junction (a junction should not have the same image as a station so users can distinguish the difference through the GUI). As a result, unnecessary methods for drawing the *CollisionStationActor*s separately from StationActorswere removed to clean up the code to improve coding efficiency, increase the code quality and lower the amount of processing required by iterating through the list of stations only once as opposed to twice.

Additive Changes

Track Modification

The track modification feature was the first of two prioritised requirements to implement into the TaxE game within the current development stage(User Req Modify.2.1 , User Req Modify.2.2 , User Req Modify.2.3.). The feature will allow users to dynamically alter the maps routings between cities by allowing the addition and removal of tracks, adding a more immersive and replayable experience for the user because the game can alter however the users desire.
To implement the track modification feature, new classes were added with modifications to some existing ones. A public class *MapController* containing all logic related to any map modifications, i.e. adding and removing track connections between stations. When instantiated, the *MapController* class will validate if the *GameState* is in the newly defined state of *EDITING* when a station is clicked (*StationController* class deals with this). If false, nothing

happens and the *MapController* class exits. If true, the subroutine *editConnection(station)* is called with the parameter *station* indicating the station that was selected by the user.
The subroutine *editConnection(station)* is called when a station is clicked and will determine whether to remove or add a connection, calling the respective method to execute. Firstly, the subroutine will check whether or not the *station* selected is not stored in the *origin* variable or if *origin* does not contain a current value. If *origin* contains no value or already contains the *station* specified, then store *station* in *origin*. This ensures that one *Station* (node) is selected before attempting to remove a connection, otherwise the program will crash from attempting to remove a connection from a null value (i.e. the middle of nowhere!). If a *station* different to the *origin* has been selected and *origin* already contains a *Station*, then assign the new *station* to the *destination* variable to indicate the other *station* (node) to relate to. A boolean variable of *success* is created to use as an indicator later on to determine whether or not a connection was successfully modified.
Afterwards, the subroutine will check if a connection exists between the two *station* variables, *origin* and *destination*. If no *connection* exists, the *addconnection()* function will be called and allocate its determined value to the *success* variable. If a *connection* does exist, the *removeconnection()* function will be called and store its determined value to the *success* variable. These two subroutines allowed us to split the coding up to comply with the clientele's requirements that users must be able to add (Func. Req Modify.2.2) and remove connections (Func. Req Modify.2.3 , Func. Req Modify.2.4) from the map.
The *addConnection()* subroutine (Func Req Modify.2.2) will firstly make the *connection* between the *origin* and *destination* stations, draw the *connection* on the GUI and check whether it overlaps/intersects with a current connection. A for loop goes through every *connection* currently in-game (*existingConnection* in *context.getGameLogic().getMap().getConnections()*) and validating whether it intersects with an existing connection (*connection.intersect(existingConnection))*. This has to be when one *station* is not equal to *destination* and the other one is not equal to *origin* as otherwise the function would return *false* on a viable *connection* and prevent it from occurring in the game, which would indicate a bug in a primary feature of the game, potentially making users and the clientele unhappy. If the new *connection* does intersect with an *existingConnection*, then the *ConnectionModifier* resource is not consumed and an error message is displayed at the top of the screen to the user to indicate the *connection* cannot be made. It was necessary to not consume the *ConnectionModifier* resource to stop the user getting frustrated over losing a resource at a track modification they performed by accident or had been unaware of its restrictions and therefore losing their advantage in the game for performing better (Func. Req Modify.2.1). Afterwards, the subroutine will create a new *Actor* (symbolises new *connection* made), add the *connection* to the *Map* (make it visible on GUI to user) (User UI Req Modify.5.2), add the *Actor* to the stage (symbolises it exists in the game and program knows), add *Actor*s for *stations* again (puts them back on top to allow stations to be drawn in front of the connections for UI purposes as users click stations the most) and finally add all *train Actor*s again (puts them back on top too for same reasoning). As this was the way the architecture was set up, it was the only possible way the developers could realistically create the feature with the least amount of inconvenience without investing large amounts of time unnecessarily. Additionally, the reason for implementing the intersection check is to remove the need to create a new *junction*. The decision was made by our team to reduce the amount of complex coding required as the team was limited with the amount of time and resources available and did not want to make coding more difficult than it currently was based on the system architecture provided. It was also the preferred method to construct the *addconnection()* routine on based the team's prior knowledge and the support available and did not state in the requirements or by the clientele that new junctions or cities had to be explicitly added, only suggested, which meant that this was on a lower priority for this development stage.
On the other hand, the *removeConnection()* (Func Req Modify.2.3) will remove a *connection* if no *Train* is present on the track and alter a *Train*'s route to comply with the *connection* removal by making the new *destination* the final *station* before the removed *connection* point. The justification behind this is that the gameplay experience may have been reduced or become more frustrating for users when a *Player* in the lead has a greater chance of staying in the lead by removing their opponent's *Train*s through track modification. From a developer's perspective, it removed the need to remove the *Train* in question from the *Player*'s *Resource*s and the GUI, helping to reduce complex coding that could have been hard to understand or implement in which large amounts of time would have been invested.
Firstly, to implement this, the subroutine will assign the *train* variable as type *Train* containing the *Train resource*. Afterwards, it checks with an if statement whether the *connection* in question contains a *Player*'s *Train* by running through two for loops (one to go between each *Player*, another to go through each *Train*) that checks if a *Train* is between two stations, *origin* and *destination*, by seeing which *Station*s the train has recently passed. This is done by using the newly defined methods in the *Train*, *getMostRecentlyVisitedStation* and *getNextStationOfRoute*, to assign the the two *Station*s to *origin* and *destination* respectively. If a *Train* is present on the *connection*, return *false* to indicate a change cannot be made with an appropriate error message to the user in red font (same reasoning behind this error message as previous error message). If not then the subroutine will check for any *Train* using the *connection* as part of their route and will alter the *destination* station of the route to the *origin* of the

removeConnection() station (i.e. the first one) if the *Train* is moving. This is because a *Train* will not need its route to be altered if it is stationary, having no interaction with the *connection* and would have caused unnecessary processing and potential crashes through clashes with the code. In addition, the altering of the route prevents a bug occurring where a *Train* would have attempted to use a *connection* that does not exist, meaning an increase in the software's quality through readability and maintainability. At the end, the subroutine finds the *connection*'s *Actor*, removes it from the *Array<Actor>* list (if it exists) to remove it from the game's view (GUI) to visibly show it does not exist to the users (User UI Req Modify.5.2), and removes it from the game's logic so that a non-existent *connection* may not be called that could otherwise crash the program. The subroutine will verify whether the connection was successful (*success = true*) and if so, switch the *GameState* back to *NORMAL* because no more connections will be modified and gameplay can resume as normal. Finally, to stop users repeatedly using
In addition, a *Player*'s *Goal* would only generate for accessible stations where the *Goal* would be completable (Func. Req Modify.2.5). This is because the *origin* and *destination* stations would not be connected by any sequence of connections visibly on the GUI and in the system. As a result, a *Player* should not feel restricted in having to wait until a connection is added back in to reach their desired goal as all their goals might have this condition and hence cannot complete any goal. The user's game experience would be lowered and make the game less competitive throughout if one *Player* had reached this point because the other *Player* could easily win. It would also prevent the user from consistently abandoning goals until one was achievable as many turns could pass until the user could return to playing the game and complete goals. This mechanic was not too hard to implement and did not take up considerable amounts of time and would increase the gameplay experience greatly, therefore it was recommended to the clientele by the team in which they agreed.
As a final constraint, track modification did not want to occur too often in the game to disrupt the gameplay too often and needed to be given at a reasonable point in time to ensure it is used, therefore the team decided it would act as a *Resource* and would only be allocated to a *Player* once they completed had completed a goal (Func. Req Modify.2.1). The reason for this is to limit the amount of modifications that can occur on the map, as otherwise the map would have been constantly changing and the core game mechanics of completing goals would be near impossible as a *Train* may never reach its *Goal* or constantly change its route. As a result, the users have a reduced game experience, hence find it boring or not as fun and not play the game anymore. To implement the constraint, a new class *ConnectionModifier* was added that extends the *Resource* class to give ownership of the charges to a *Player*, similar to that of the *Resource*s previously used for giving *Player*s a train and ownership of it. This saved time for the developers in using the available classes to prevent the need to setup repetitive code and to keep the system consistent and maintainable for any future developers. The game also needed to add a charge when a user completed a goal, hence the *Player* class was altered to add the track modification resource (*addResource(connectionModifier)*) to the *completeGoal* routine. The user has no limit to the number of track modification charges they have as long as they complete the goals to acquire them. This is to allow a user to store the resource for future use and have the flexibility and freedom to use them whenever they wish, which makes a dynamic experience every time the user plays the game.

GUI
The GUI allows the user to initiate the track modifications through a button in the top-right corner (User UI Req Modify.5.1) [Appendix N, 1]. The button will only become visible when a *Player* has a *ConnectionModifier* resource to spend because without it, the *Player* would not be able to make a track modification therefore there is no point in having the button visible. Users may otherwise believe they have a track modification resource to spend which may lead them to believing the game is bugged and put them off playing the game. The button's colour scheme and font follow the theme throughout the game, being consistent with the other buttons by using a white font against a grey background. The contrast in colours against the background help the clarity of the button's text for the user to increase readability and look professional. The button also indicates the number of track modifications available to the user to provide transparency and track how many they have available to use if they wish to reach a certain amount or know how many modifications they are allowed to make. By being located in the top right corner of the GUI, the button can be clearly visible to all users and allowed it to not clutter up the main game board for the users, separating the game's buttons from the game itself. It was also one of the easiest areas to place the button for the developers, due to the internal structure and already implemented classes for performing this, and meant that the GUI had spare space on the game board for further features in the future that directly impacted the game board itself (meant GUI was more flexible and extendable).
From this, the user enters the track modification screen where a reduced HUD is shown (i.e. hidden buttons at top of the screen) whereby users indicate the track to add/remove by clicking on the two stations that the track connects to (User UI Req Modify.5.2). The developers did this to prevent any subroutines trying to process over each other that may otherwise crash the game (e.g. trying to replay the previous moves and perform track modification simultaneously) and reduce the amount of processing required to the screen where it can be done as the other

buttons are unnecessary here. It was also the simplest method to implement the adding and removal of tracks. From a user's point of view, it allows them to clarify which mode they entered, which buttons are not necessary to use and prevents them from accidentally performing a different action to improve gameplay experience. Also, it was viewed as a simple mechanic for users to perform through clicking the stations for an increased gameplay experience by not being complicated to perform (good for new users of the game).
As stated earlier, an error message is also displayed at the top centre of the screen in a red font whenever an invalid connection was attempted. A red font is used to show it is an error or invalid and to increase readability through a contrasting colour against the background.It also had to display an error message to make sure a user did not repeatedly try to make the same *connection* in believing it is viable as they may believe the game is broken when it is not (User UI Req Modify.5.3). By being located at the top centre of the screen, the message can be shown throughout all possible menus and to make users focus upon it more as generally users read information from the centre or top of the screen (here, it would be the top of the screen to reduce UI cluttering at the centre and annoying users by being directly placed on top of the game board when they are trying to play). The message is shown in the key information area to show it has importance in the game.

Tests
Unit test cases through JUnit were added to verify and validate the track modification mechanic. The following unit tests were added and all passed:

- intersectingConnectionsTest (Unit Test ID 1.1) - detect that two connections intersect within the map
- nonIntersectingConnectionsTestOne (Unit Test ID 1.2) - detect that two lines (despite not being parallel) do not intersect within the map
- nonIntersectingConnectionsTestTwo (Unit Test ID 1.3) - detect that two parallel lines do not intersect within the map
- testAddConnectionModifierToPlayer (Unit Test ID 2.1) - detect that if a player is given a connection modifier, it will be contained within their resources.

Further evidence of testing for this feature can be found in acceptance testing scenario 2 [19] and the system testing log [20].

Replay System
The replay system feature [Appendix N, 2] was the second of two prioritised requirements to implement into the TaxE game within the current development stage (User Req Replay.1.1 , User Req Replay.1.2). The feature will allow the system record the state of the Game frequently while playing, and also allow the user to click a button to start a Replay of the last sets of actions. While replaying, the system quickly restores these recorded states, called snapshots, and shows them back at a given interval .The 'replay' mode can be played back at a faster than normal pace if desired, at 1x, 2x, 3x, 4x or 5x the speed. The idea of the feature is to increase the gameplay experience by allowing users to go over past moves in the game where they may have missed their opponent's move or they wish to view a move again to help them strategically plan their next move.
To implement the replay system feature, the system needed to implement a method in capturing each state of the game. As a result of research, a developer discovered the best way to do this was by using "snapshots" (Snapshot class) that stored a particular state of time in the Game. The class utilises the Apache Commons Java lang for extra utilities (import org.apache.commons.lang3.SerializationUtils) and was defined with all the appropriate properties that wished to be stored to enable each snapshotted state to store recent events appropriate, such as resources used or gained on each turn, the Trains of each user and their location, etcetera. It extends a Serializable interface to allow it to be written to and store in many locations (hard disk, a database, a network socket and so on). This allows the system to be extendable with ease if the data wishes to be stored in some other form in the future, e.g. allowing the game to be played on a network for remote playability with anyone in the world to increase the target audience range.
The feature works by running through the createSnapshot method continuously while in game, every time the State of the Game changes. The method creates a new *Snapshot* data structure containing the playerManager (handles both Players), goalManager (handles goals for each Player), resourceManager (handles resources for each Player), obstacleManager (handles Obstacles in the game), map (the game map for this class instance), state (the GameState currently in motion), confirmingTrain (The train that is selected for routing) and confirmingPositions (The position the train should be routed to). Each is assigned a variable value based on these at each state of the game when a screenshot is captured, along with the *players* (*b.getPlayers()* to retrieve number of players in game), *currentTurn* (*b.getCurrentTurn()* to retrieve which players turn it is) and, *turnNumber* (*b.getTurnNumber()* to retrieve the number of turns currently passed). The routine will then continuously add snapshots to the game, capturing continuous states of the game. If no exceptions were caught, the program will have the snapshot stored in an internal list and print the number of current snapshots (*getSnapshotNumber()*) currently in memory.
The replay system is started when the "Replay" button is clicked, executing the *replaySnapshot(int*

snapshotNumber) where *snapshotNumber* is an integer value that represents the number of snapshot stored in memory to replay. Initally, this is set to 0, in order to start the replay from the beginning -the first snapshot-. The aim of the subroutine is to navigate to different points in time by traversing each *Snapshot* in chronological order. If a *Snapshot* point is given in the past, then enable Replay Mode. If the *Snapshot* reaches the current point in time, then disable Replay Mode to indicate to the system to resume normal gameplay.
Initially, the subroutine will store the snapshots in variable *s* of type *Snapshot* and run through an exception handler that deserialises each *Snapshot* stored in memory. Afterwards, each snapshot's contents will be loaded into the current game using *loadSnapshot(Snapshot s)* where the *s* parameter is each snapshot being passed as an argument. This routine is to enable the *replaySnapshot* method to retrieve every snapshot in order to allow the replay system to execute the events in chronological order and with the correct event information. An update of the game interface is forcefully triggered (*stateChanged()*) to notify the *GameState* listeners that a change occurred to the games state, i.e. a snapshot was loaded. Some of the GameState listeners that are responsible for updating the interface, will notice that the Replay Mode is enabled and will modify the interface accordingly, hiding excess buttons and showing the current progress in the replay.
The program returns to *replaySnapshot* and verifies that the game is still in replay mode (*replayMode = true* if the *snapshotNumber* is not equal to the number last snapshot, which indicates that there are still snapshots to be replayed to the user) to stop the program continuously being in Replay Mode so that users can return to playing the game afterwards. If *replayMode* is true and enabled, then the game will acquire the snapshot number to play next by incrementing its internal counter. Some code implemented into the rendering task of the Game counts the time passed between each frame, using an internal counter and a threshold in order to create a simple timing mechanism. This was the simplest and most code efficient way of implementing a Timing mechanism as it does not need to create a new Timer Thread that would not be able to easily interact with the UI context and draw to the interface because of some OpenGL limitations. The timer, every 250ms, will check if the Replay is idle -i.e. not animating a train- and will eventually load the next Snapshot in memory. One final check validates whether or not *replayMode* is still active and, if it is not, will set the game's speed back to normal (*getInstance().setGameSpeed(1.0f)*) and display the obstacles (*ActorsManager.showAllObstacles()*) to allow the users to resume the game from the point at which the game entered replay mode, otherwise gameplay experience would be reduced and users would become frustrated by not being able to continue from where they left off.
As a requirement recommended by the clientele, the team implemented the option to view the replay at varying speeds which allowed the user to control the pace that they wished to view the replay in. This gave the user more freedom over how they viewed the replay to help them strategically their next move and meant that they could return to the real-time action as soon as possible. The replay speed could be adjusted using a slider that set the game speed (*setGameSpeed*) based on a provided *speedMultiplier* value that altered the game speed accordingly.

GUI
The GUI allows the user to enter the replay system through a button located in the top right hand corner (User UI Req Replay.4.2) [Appendix N, 2]. The reasoning behind the use of a button and the location on the GUI is the same as the explanation in the track modification section (see previous explained additive change). The button also indicates the progress of the replay as a percentage (based on number of snapshots it has gone through) when in the separate replay mode so that users know how long is left of the replay before they can go back into the normal game.
The replay system does, however, incorporate a slider (User UI Req Replay.4.1) to alter the replay system's speed. A slider allowed users to quickly alter the speed by dragging the slider along to the appropriate speed point they wanted or by a simple click on the slide bar. This was a design decision as most games used this method to alter settings such as volume and seemed the most logical method of implementing the feature based on the team's and clientele's preferences. In addition, the slider is only visible when the mouse hovers over the replay button to hide the slider when not in use, preventing the users from accidentally changing the slider value from the one they want. It also meant the UI was not as cluttered to increase the quality of the UI and stop the user feeling overwhelmed by many buttons being on the screen at once which may otherwise have put them off playing the game.

Tests
Unit tests cases through JUnit were added to verify and validate the replay system mechanic. The following unit tests were added and all passed:

- testSnapshots (Unit Test ID 3.1) - snapshots are taken at correct intervals and with correct number
- testSetReplaySpeed (Unit Test ID 3.2) - replay speed can be altered up to a maximum of 5x normal speed.

Further evidence of testing for this feature can be found in acceptance testing scenario 1 [19] and the system testing log [20].
Mobile Port / Cross-platform functionality

Developing the game onto a mobile/tablet operating system [Appendix N, 3] was a lower prioritised, preferable requirement to implement into the TaxE game within the current development stage (User Req CP.3.1 , User UI Req CP.6.1 , Func. Req CP.3.1). The feature will allow a version of the game to run on the Android or iOS mobile operating systems, increasing the portability of the system and hence increase the target audience range for the clientele's product as some users may prefer to play games on their mobile device rather than a desktop or laptop. It was recommended by the clientele to include once put forward by our team as a possible extension to the game.
The initial plan was to ensure a mobile version of the game was possible with relative ease as no team member had made an Android or iOS game before. One of our development team researched into the possibility of porting it. Based on their findings, the team found that the previously used framework in the game, libGDX (previously explained), can make it happen. LibGDX has cross-platform functionality whereby a project using Gradle can build a project for desktop, Android, html or iOS. Also the resolution fix, which was discussed earlier, would allow the game to run on the smaller screen of a mobile device.
After discovering the possibility of cross-platform functionality, our team decided to build ADB's TaxE game version (from assessment 3) as a testing point. A new libGDX project as a separate GitHub repository was created with the Desktop, Android and iOS subprojects chosen to be included (html could not be included because it did not support some of the librarys used for the game). This was to stop any interactions with the currently working version in the event that the functionality of the TaxE game might have been affected by any changes in the cross platform project while the other developers were working on the main project. As a result, the reliability of the project and the chance of being bug free was increased and hence the software quality.
As a new GitHub repository, the same code base could be used in the cross platform project, except that it will also run on Android or iOS once it has been compiled, hence the code only had to be transferred across to the cross platform repository. By utilising the same resources, considerable amounts of time were saved to implement a preferable feature for the clientele, hence it was possible to implement within the timeframe without any project delays. It also showed our project was highly maintainable and extendable for the future to make it port to many different operating systems. As a result, the mobile TaxE game has the same functionality as the desktop version to keep consistency between all devices with their implemented features which can only improve the target audience range and customer satisfaction because they may choose a wide range of devices to play on and allows the game to be portable.

Tests
Unfortunately for the testing, only the Android version was tested because the game could be downloaded to a device using android development tools for eclipse for free while an iOS version of the game had to be downloaded after uploading the application to Apple's App Store. Uploading the game's application to the App Store required a payment unless the iOS device had been jailbreaked (removing Apple's hardware restrictions [18]). In most countries, it is against the law to jailbreak an iOS device, therefore the team made no attempt in jailbreaking an iOS device incase one of the team or the clientele ended up in court. The game did, however, prove to work on an Android device, hence it may be assumed that the game also works on an iOS device. Evidence of testing can be located in the system testing log [20].

Cancelling Goals
During previous implementations, the TaxE game had been modified to include absolute and quantifiable goals. These have been implemented to their specified requirements, however, the team noticed that the goals could not be cancelled at any given point and therefore was considered to be a preferable requirement when put forward to the clientele [Appendix N, 4] (User Req Goal.4.1 , Func. Req Goal.4.1 , Func. Req Goal.4.2). The feature would allow the user to keep the goal objectives they want to keep and remove those they do not want, allowing them to have more freedom in what goals they achieve, which can only mean a more dynamic and interesting game experience. The button used (User UI Req Goal.7.1) depicts a large X icon next to each goal to distinctly indicate that, once clicked, which goal would be removed to give the user transparency and clarity over the goal they are about to remove. This should act as a clear indicator to prevent users removing the wrong goal by accident and stop them becoming frustrated. A red background also helps to clarify the button area for the user to click by contrasting this background colour against the game board's colour. A notification pop up (User UI Req Goal.7.1) to verify that the user wants to delete the goal prevents the user from accidentally removing the goal at any point during the game to improve the gameplay experience and stop them becoming frustrated if they performed this on a goal they wanted to complete. It is placed at the centre of the screen to ensure users see the notification through this focus as, generally, this is the area users will view the most. It was a design decision suggested by the team and clientele to implement.
Evidence of testing can be found in acceptance testing scenario 1 [19] and the system testing log [20].

References

1. Gradle, “What is Gradle?”, Internet: https://gradle.org/ , [Feb. 11th 2015]

2. Wikipedia (from Mobile Game Engines), “LibGdx”, Internet: http://en.wikipedia.org/wiki/LibGDX (http://mobilegameengines.com/game_engines/32libgdx), [Feb. 11th 2015]

3. JUnit, “About JUnit”, Internet: http://junit.org/ . [Feb. 12th 2015]

4. Oracle, “JavaDoc Tool”, Internet: http://www.oracle.com/technetwork/java/javase/documentation/indexjsp135444.html , [Feb. 12th 2015]

5. M. Fowler, Refactoring: Improving the design of existing code , Boston; London, : AddisonWesley, 1999.

6. TravisCI, “TravisCI Free Hosted Continuous Integration Platform for the Open Source Community”, Internet: https://travisci.org/recent , [Feb. 13th 2015]

7. Team ADB’s architecture report

8. Team ADB’s requirements specification from Assessment 1

9. D. Parnas, P. Clements, “A Rational Design Process: How and Why to Fake it”, IEEE Transactions on Software Engineering, 1986, pp. 251257

10. Assessment 4’s brief

11. Team EEP’s test plan

12. S. R. Pressman, Software Engineering: A practitioner’s approach, Boston ; London : McGraw-Hill Higher Education, 7th edition, 2010.

13. SCM Wise, “Software Change Management”, Internet: http://www.scmwise.com/software-change-management.html , [Apr. 2nd 2015]

14. I. Crnković, U. Asklund, A. Persson-Dahlqvist, Implementing and Integrating Product Data Management and Software Configuration Management, London: Artech House, 2003

15. University of York Computer Science Department, “SEPR Requirements Changes”, Internet: http://www-module.cs.york.ac.uk/sepr/RequirementsChanges-201415.html , [Feb. 23rd 2015]

16. K. Cassidy, J. Dries, “PNG: The Portable Network Graphic”, Informit, Internet: http://www.informit.com/articles/article.aspx?p=23959&seqNum=4, Nov. 2nd 2001 [Apr. 4th 2015]

17. FVS, “FVS Architecture Report”, Internet: http://www-users.york.ac.uk/~oeh503/fvs/ , [Feb. 23rd 2015]

18. Wikipedia, “iOS jailbreaking”, http://en.wikipedia.org/wiki/IOS_jailbreaking , [Apr. 4th 2015]

19. Team EEP’s Acceptance Testing

20. Team EEP’s System Testing Log

21. Team EEP’s Assessment 1 Requirements Specification