Change report

Approaches

Our team's initial approach to change within the project was to keep our changed documentation and code separate from the original. We did this by forking their code on github to our own team directory, and by retaining team Muscovy's documentation and making any changes in separately saved documents. A reason that we did this was to allow better tracing of changes; it was clear and easy to see where changes were made when it was possible to compare side-by-side with the original. Github was very useful in helping us to retain the original code, along with previous code before commits were made. This also allowed for a fall-back working implementation in case any changes introduced new bugs, or files became lost/corrupted, etc.

This was also useful when assessing how well our changes helped us to fulfil the requirements specification. The increased focus on traceability helped us to follow how our changes related to the completion of the requirements. Therefore, it let us see how our further implementation directly lead to completely fulfilling team Muscovy's requirements. This allowed us to clearly show justification for our decisions with regards to change in the code, as we could easily refer it back to what had not yet been implemented from the requirements specification.

Our team's approach to actual implementation of changes used the 'change-request' format, with minor changes. By 'change-request' format, I am referring to the handling of change whereby personnel involved with the program may request (propose) changes [1]. In a larger-scale, typical industry project, there are more personnel which may propose changes - for example, users, designers, system analysts, and so on [1]. The requested change is then evaluated by a Change Control Board - in this case, our team as a whole - based on cost, and benefit to the project. Following this, it is either accepted or rejected, and given a priority [1]. The process is shown in diagram form in Fig 3 [1]. As a smaller-scale project, we handled change proposal itself slightly differently - we didn't feel the need for a change proposal form, or similar, due to the tight nature of our team. Instead, initially, changes were proposed through the 'Issues' functionality in github. These were organised by urgency, with requirement fulfilling being the most urgent, followed by bug-fixing, and enhancement (perfective) changes being the least urgent. This provided a list of proposed changes which we could easily review as a team in order of importance - shown in Fig 2. To better illustrate the changes needed, during review, changes such as 'needs more objectives' were also refined into concrete goals and tasks .

We knew that our changes would cover the range of perfective, corrective, additive and deletive changes. Therefore, we devised slightly different protocols for each type of change, due to the perceived difference in impact and risk associated with each one. The process associated with the changes we made was also often based on the estimated severity of the change in question. To elaborate, corrective changes which were thought to be very minor – e.g., changing misleading variable names, or fixing typos/misspellings in documentation – were left to the discretion of the team member carrying them out. Corrective changes as a whole were less scrutinised than other types of changes. As long as the aspect of the software/documentation was agreed to be a fault by the team, the corrective changes were carried out as needed.

Perfective, additive and deletive changes were agreed upon by the team beforehand. Perfective and deletive were given less priority than additive changes; this was due to the fact that additive changes were more likely to directly fulfil an aspect of the requirements, whereas perfective changes often only optimised content which was already implemented. We did not expect to have many deletive changes and did not plan extensively for them; our goal in assessment 3 is to build on the code and documentation which we received from team Muscovy. As their implementation has been the basis/core functionality of the game, there would not be many necessary deletive changes. In the documentation, however, they were occasionally necessary – for example, when editing documentation which had explicitly referred to previous team members by name. These were handled individually.

Implemented Changes Report

This report is organised into sections where changes to software are iteratively covered, followed by any documentation changes resulting from the changes to the code. Changes are organised into additive, deletive, perfective and corrective sections.

Direct download links to documentation are referenced throughout the document. However, the pages where the documentation is collected are as follows:

Swapped files from previous team

http://teal-duck.github.io/teal-duck/swapfiles2.html

Our Assessment 3 documentation

http://teal-duck.github.io/teal-duck/a3files.html

Additive

Software change

In the projectile class, a 'shootProjectiles()' static function was added. This was a function which handled projectile shooting and also provided functionality for bullet spread in the case of multiple bullets (e.g. for bosses, or when the player acquires the triple shot power-up). This avoided duplication of logic as it could be used by both the player and enemies/bosses. This made fulfilling requirements 5.2, 6 and 8.1 (https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/req2.pdf) more efficient with regards to how much code was needed; handling any number of bullets firing in one function allowed us to easily implement power-ups, and unique (objective- or round-specific) bosses.

Subsequent documentation change(s)

References to the previously planned function intended to handle projectile firing, 'OnHit()', was removed from the new architecture document.

Software change

Resource pickups were implemented to fulfil requirements 8.2 and 9 (https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/req2.pdf). These resources had to be, as specified in the requirements, useful to help the player to finish the game. Therefore we added health pickups as well as power-ups which increased the power of the player's weapon; bombs were also implemented as a power-up, but the bomb resource worked the same as health in that the player could only have a limited number of bombs, which could be increased by picking up a bomb item. This required the addition of an 'ItemType' class containing the ItemType enum and texture locations. Additionally, item handling logic was added to the class EntityManager.

Subsequent documentation change(s)

The class for handling resources had already been shown in the previous team's architecture as 'Collectible'; however, theirs inherited from the scrapped class 'Destructible', which was a subclass of 'Obstacle'. This is not how we implemented resources and pickups; we implemented an Item class which is a subclass of Collidable. Instances of this class as they appear in the game have an associated ItemType enum which decides which logic is carried out with regards to the item. This information is shown on the new class diagram in the modified architecture report at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf and in the appendix of this report.

Software change

A class MoveableEntity was implemented to share moving logic between the player and enemy entities, by collecting accessors like getVelocityX, getVelocityY, getFriction, and mutators setVelocity, setAccelerationSpeed, and so on. This made it exceedingly simple to implement the controllable player, as detailed in user requirement 15, and fulfil functional requirement 5.5 (https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/req2.p df), which states that ranged enemies may move, and may not; this is done by varying the velocity parameter. The fact that the logic is shared increases efficiency with respect to code size, and follows the general principle of reusing code where possible.

Subsequent documentation change(s)

The class MoveableEntity and its inheritance relationships are shown on the updated class diagram. This class extends Collidable, as shown on the diagram in the appendix of https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf and of this report. It allows the player, enemies and bombs to share the same movement logic, as they all inherit from the MoveableEntity class.

Software change

The flight and swimming inherent abilities were added to fulfil requirement 8.1. Flight was implemented by allowing the player to increase their speed for a limited length of time at a time; we did this by adding flight speed multiplier and max flight time variables to the PlayerCharacter class. Swimming was also implemented through a flooded level (Goodricke) where the player's friction is increased.

Subsequent documentation change(s)

This added functionality was implemented through parameter changes to classes and so didn't require representation in the documentation.

Software change

A full suite of acquired player abilities were added to the game to fulfil requirement 8.2. The player now acquires a new power-up upon completion of each level. Power-ups appear as pickups, and upon collection by the player, they alter the associated enum(s) PlayerShotType and ProjectileType, or attributes associated with the PlayerCharacter class such as maxHealth. This allows us to vary the player ability as they progress through the game. The abilities we added are: Triple-shot, Rapid fire, Flamethrower, Flight, Bombs, Extra Health and Sunglasses.

Subsequent documentation change(s)

Representation of the enums used to implement some of these abilities was added to the new class diagram found at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf. Compare with the previous team's architecture at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/arch2.pdf, where special abilities were unclearly labelled as an 'Ability' class inheriting from 'Collectible'. In our representation, ability pick-ups are instances of the 'Item' class but, as previously discussed, are handled using enums and attribute changes.

Software change

Implementing saving and loading functionalities. Requirement 2 and 2.1 (https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/req2.pdf) state that the game should be saveable and loadable from the current state, and that it should save after each level completion. This was not implemented to any level when we inherited the code; upon completion of the game, we have fully implemented it as specified in the requirements.

Subsequent documentation change(s)

By inheriting from the abstract class BaseSerializer and using the interface provided by JSON, we added multiple classes to serialize levels, each level individually, dungeon rooms, enemies, items, obstacles, the player data and save data respectively. These were then saved using a SaveGame class inheriting from the Saver superclass. This then calls the class SaveHandler when saving a game state. This information and more explanation is shown in our modified architecture document at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf. This is changed from our inherited architecture as no saving functionality was implemented or represented when we received the project:

https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/arch2.pdf.

Software change

Requirement 5.2 states (https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/req2.pdf) that the game must contain at least one objective-specific obstacle. We fulfilled this requirement by providing unique bosses for the Constantine, Law and Management and Goodricke levels. This was done by varying parameters passed to the BossParameters class in these different levels: projectile type, projectile speed, touch damage and health, among others, were altered to produce more difficult bosses in later levels.

Subsequent documentation change(s)

This change was technical rather than architectural and so did not require documentation changes.

Corrective

Software change

Many sets of variables were changed to be implemented as enums. This change was in order to allow us to fulfil functional requirements (found at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/req2.pdf) 5, 6 and 8. To elaborate, enums provided us with an easy way to implement the functionality of different objectives and different acquired abilities for the player. Our 'needs more objectives', 'needs more obstacles' and 'needs duck special powers' change requests (github issues) were met here. Requirement 8 specified that the player should have special abilities, and 8.1 specified that several of the abilities should be acquired throughout the game. By varying the enum PlayerShotType, we could implement ability power-ups which changed the user projectiles; for example, triple shot, rapid fire and flamethrower. Similarly, we could vary enums EnemyShotType to give enemies/bosses more projectiles to fire at a time, or make them homing - helping to satisfy functional requirement 11 (the game should have different levels of difficulty). We chose to use enums because they lock the range of possible values into the universe given by the enum, so are a secure method of varying entity parameters.

Subsequent documentation change(s)

Representation of our added enums with some example values were added to the class diagram in our modified architecture diagram, found at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf. Explanation of all enums used were added to the textual architecture report. Compare with the inherited architecture document at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%202%20swap%20files/arch2.pdf. The new class diagrams are also given as Fig 1.1 - 1.4 in the appendix of this report.

Software change

In the previous team's code, for every entity instance, the texture for that entity was reloaded from file every time it was used. This included bullets, of which there can be a large amount on the screen at a time. This method of loading textures was inefficient, adversely affected the framerate and wasted memory. The file operations were too slow to realistically use in this manner. To rectify this problem, a class TextureMap was created. All texture loading is now done through this class, which also allows disposal and overwrites of textures, so is more memory-efficient. Textures are now only loaded the first time they are requested, otherwise the already-loaded texture is returned. Additionally, all textures are disposed when the game is closed, to make sure there is no garbage left over in memory.

Subsequent documentation change(s)

This change was technical rather than architectural and so did not require documentation changes.

Software change

The enum gameState, previously held in the MuscovyGame class, was removed and replaced with libGDX screens. The enum was designed to specify the state of the game, which would then decide which screen to display throughout gameplay; screens were rendered and displayed in a single class depending on the value of the enum. Whilst being confusing, this was also bad practice as it failed to separate the screen classes sufficiently, and infringed on the class MuscovyGame by requiring logic and rendering to be done within it. To rectify this a class was created for each screen; the game state is implied by which screen the player is on, and the relevant screen and textures are rendered by the relevant class.

Subsequent documentation change(s)

These screen classes implement the interface Screen from libGDX and inherit from the abstract class ScreenBase, which holds the instance of MuscovyGame. This is shown in the new class diagram at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf, and shown in the appendix of this document. This change allowed us to meet requirements 4-4.4 of providing the user GUI, while avoiding cumbersomely rendering every screen in one class.

Perfective

Software change

The most initial changes we carried out on the code we inherited immediately were some general housekeeping and code-quality changes. These did not change the meaning or structure of the code but made it more consistent and/or readable. These changes were: removing unused variables, renaming variables to a consistent style (lowerCamelCase) and cleaning up magic numbers (giving expressive names to numbers used where the purpose and where the numbers came from was not clear).

Subsequent documentation change(s)

These changes were minor quality-of-life changes which did not affect the requirements or architecture, etc., and so didn't require changes to the inherited documentation.

Software change

The way in which boss rooms are generated in 'kill the boss' levels was changed to make it more random. This is because the previous method began from the bottom left room of the map, and picked the first room it found with only one neighbouring room. This method made it less likely that the room in which the boss would be allocated would be varied significantly in each round. Therefore the player may have found it too easy to locate the boss room after figuring out the pattern. Our new method builds a list of all rooms with only one neighbour, and randomly picks a boss room from the list.

Subsequent documentation change(s)

This change was technical rather than architectural and so did not require documentation changes.

Software change

Room templates are now loaded from a CSV file, using the newly implemented class DungeonRoomTemplateLoader. This made the code smaller, neater and more readable, as there were not large two-dimensional arrays taking up space in the room generator.

Subsequent documentation change(s)

This change was technical rather than architectural and so did not require documentation changes.

Deletive

We did not carry out any deletive changes, as the game was unfinished when we inherited it and we finished it by fulfilling the remaining requirements; therefore none of the inherited codebase became unneeded or outdated throughout our further implementation.

Other Documentation Changes

Some of our documentation changes were not made as a result of further implementation.

- Any reference to money as a resource was removed from the architecture document and class diagram. Our interpretation of requirement 9 (“The game must contain resources which the player can collect to assist them in completing the objectives.”) did not include money, but health and bomb pickups; changes can be seen at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/arch3.pdf
- The requirement for a player melee attack was removed - changes and explanation can be seen at https://github.com/teal-duck/teal-duck/raw/gh-pages/Assessment%203%20files/req3.pdf

Bibliography

[1] Enzhao Hu; Yang Liu, "IT Project Change Management," in *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on* , vol.1, no., pp.417-420, 20-22 Dec. 2008

Appendix

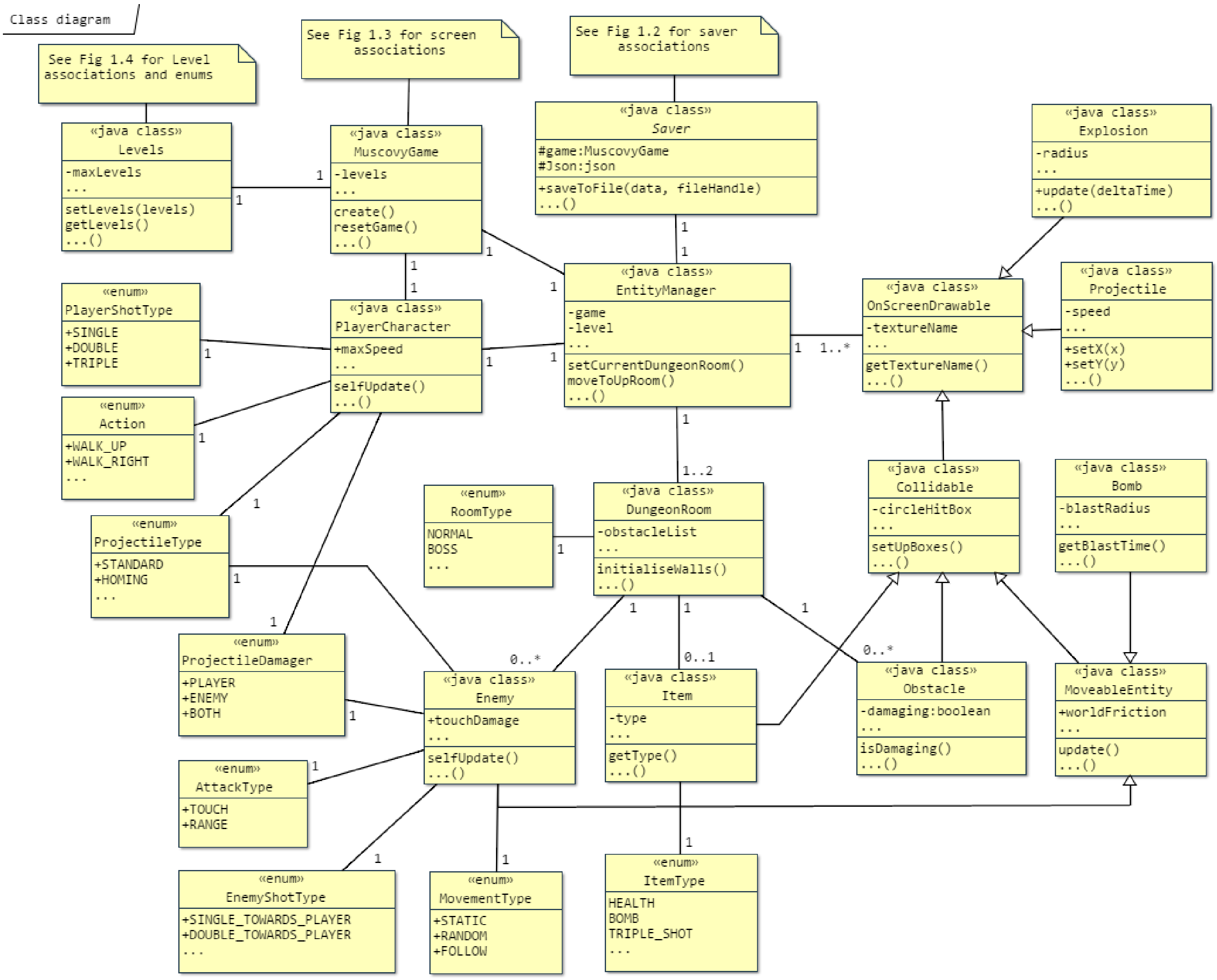
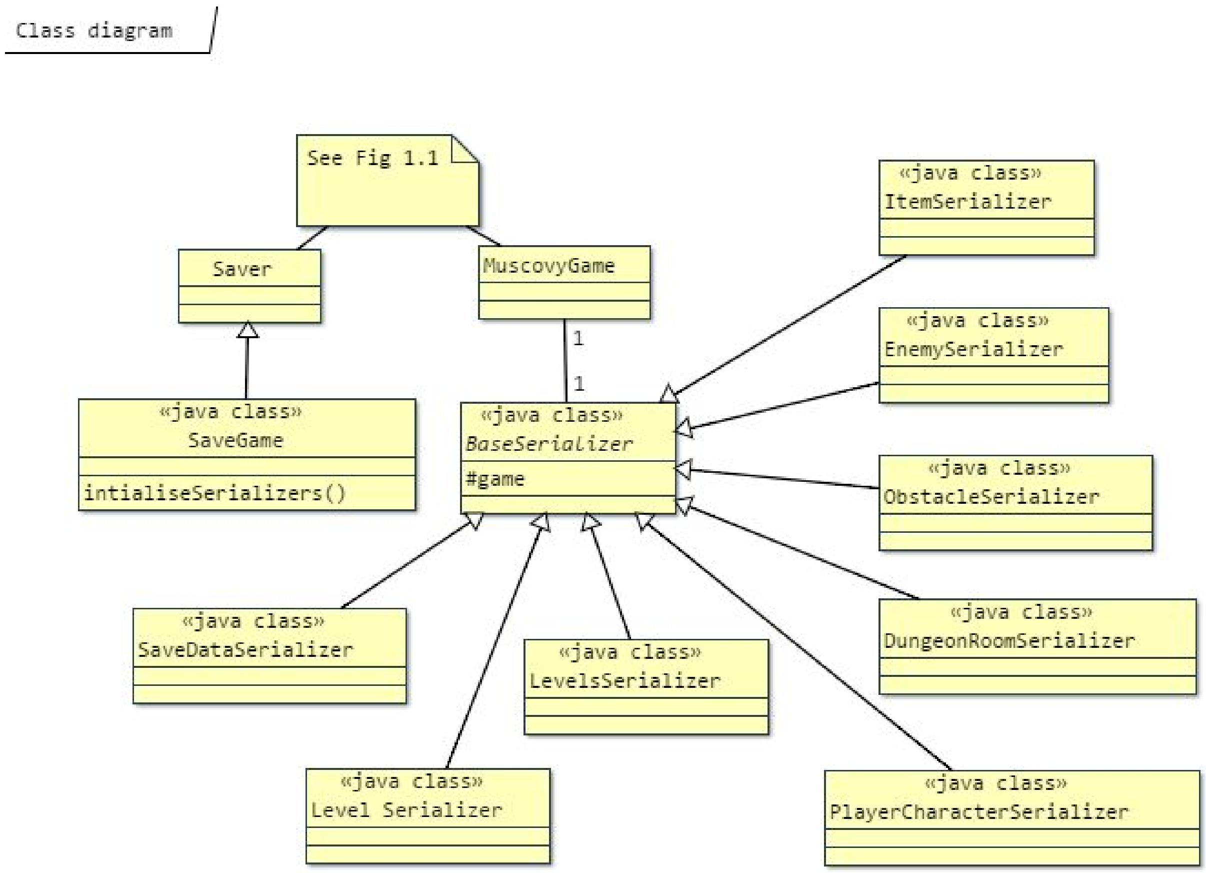
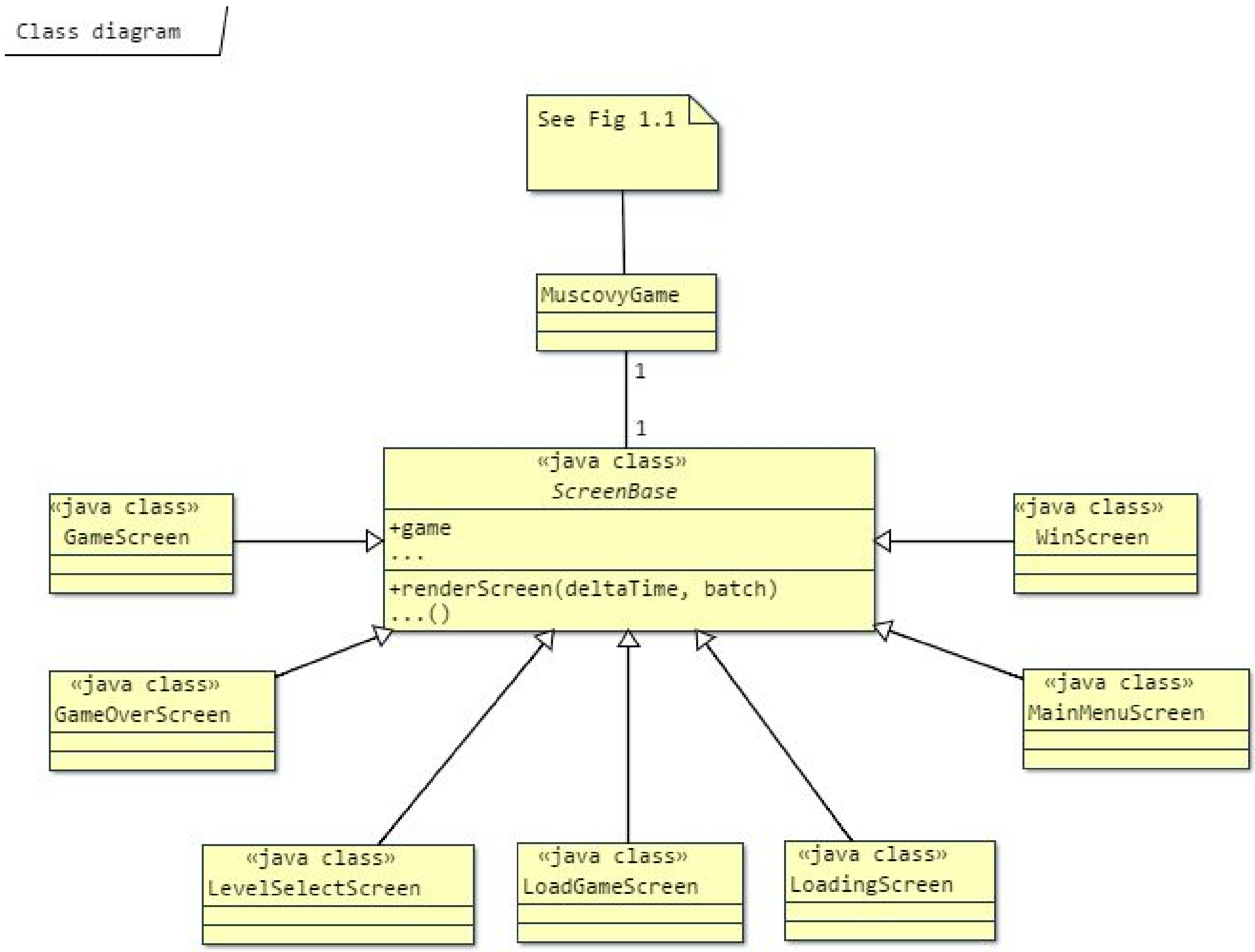


Fig 1.1. Updated Class Diagram



Class diagram
See Fig 1.1
Saver
MuscovyGame
1
1
«java class»
SaveGame
intialiseSerializers()
«java class»
BaseSerializer
#game
«java class»
ItemSerializer
«java class»
EnemySerializer
«java class»
ObstacleSerializer
«java class»
DungeonRoomSerializer
«java class»
SaveDataSerializer
«java class»
LevelsSerializer
«java class»
Level Serializer
«java class»
PlayerCharacterSerializer

Fig 1.2



Class diagram
See Fig 1.1
MuscovyGame
1
1
«java class»
ScreenBase
+game
...
+renderScreen(deltaTime, batch)
...()
«java class»
GameScreen
«java class»
WinScreen
«java class»
GameOverScreen
«java class»
MainMenuScreen
«java class»
LevelSelectScreen
«java class»
LoadGameScreen
«java class»
LoadingScreen

Fig 1.3

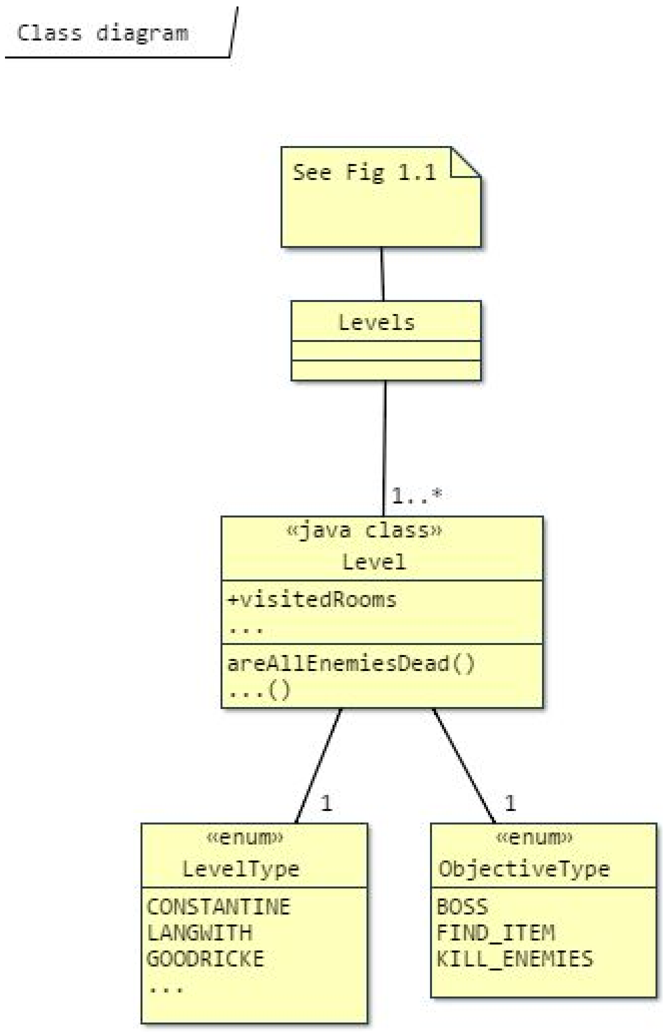
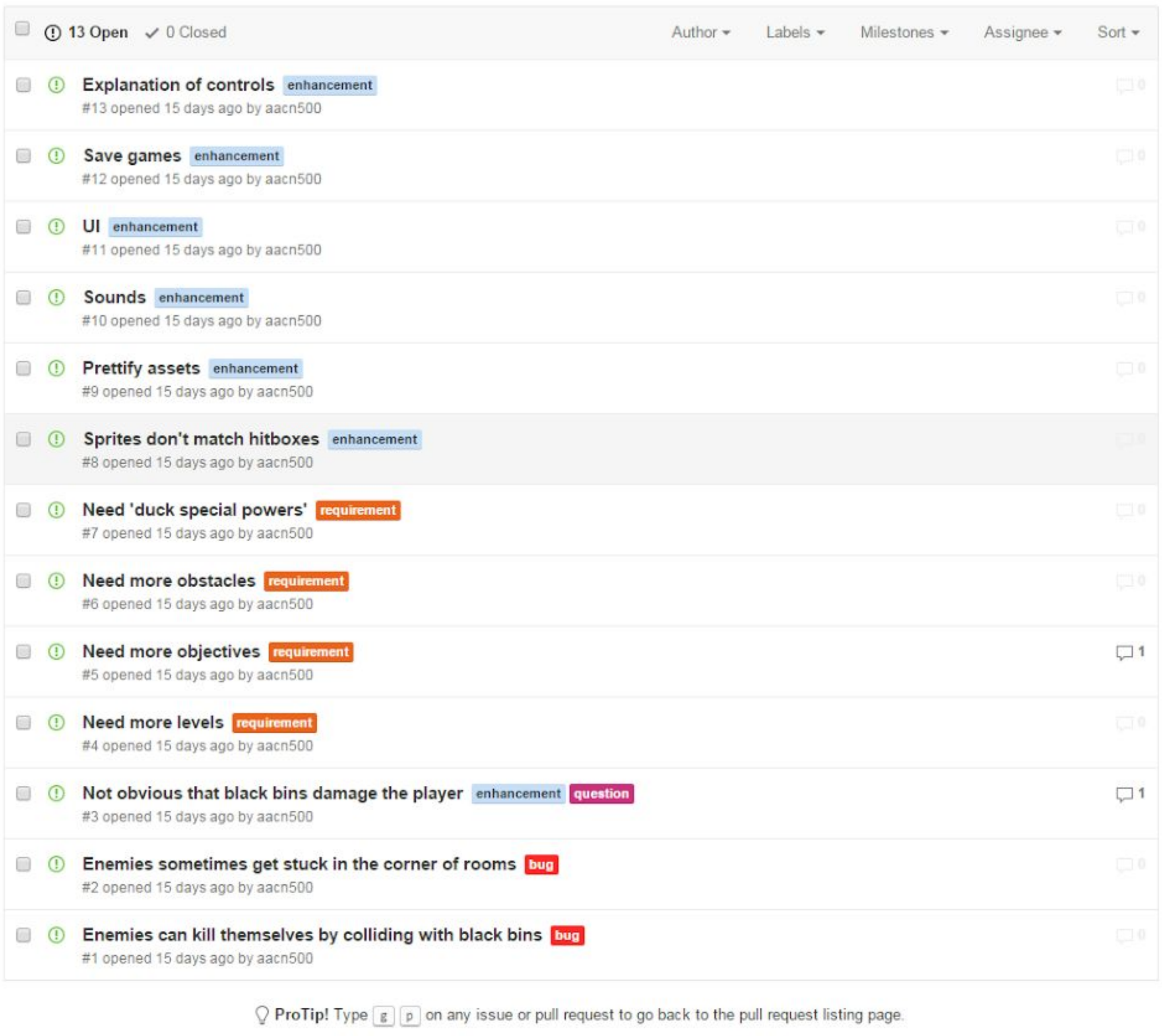

Class diagram
See Fig 1.1
Levels
1..*
«java class»
Level
+visitedRooms
...
areAllEnemiesDead()
...()
1
1
«enum»
LevelType
CONSTANTINE
LANGWITH
GOODRICKE
...
«enum»
ObjectiveType
BOSS
FIND_ITEM
KILL_ENEMIES

Fig 1.4



13 Open ✓ 0 Closed

Author ▾ Labels ▾ Milestones ▾ Assignee ▾ Sort ▾

Explanation of controls enhancement
#13 opened 15 days ago by aacn500

Save games enhancement
#12 opened 15 days ago by aacn500

UI enhancement
#11 opened 15 days ago by aacn500

Sounds enhancement
#10 opened 15 days ago by aacn500

Prettify assets enhancement
#9 opened 15 days ago by aacn500

Sprites don't match hitboxes enhancement
#8 opened 15 days ago by aacn500

Need 'duck special powers' requirement
#7 opened 15 days ago by aacn500

Need more obstacles requirement
#6 opened 15 days ago by aacn500

Need more objectives requirement 1
#5 opened 15 days ago by aacn500

Need more levels requirement
#4 opened 15 days ago by aacn500

Not obvious that black bins damage the player enhancement question 1
#3 opened 15 days ago by aacn500

Enemies sometimes get stuck in the corner of rooms bug
#2 opened 15 days ago by aacn500

Enemies can kill themselves by colliding with black bins bug
#1 opened 15 days ago by aacn500

ProTip! Type g p on any issue or pull request to go back to the pull request listing page.

Fig 2. Github Issues Screenshot

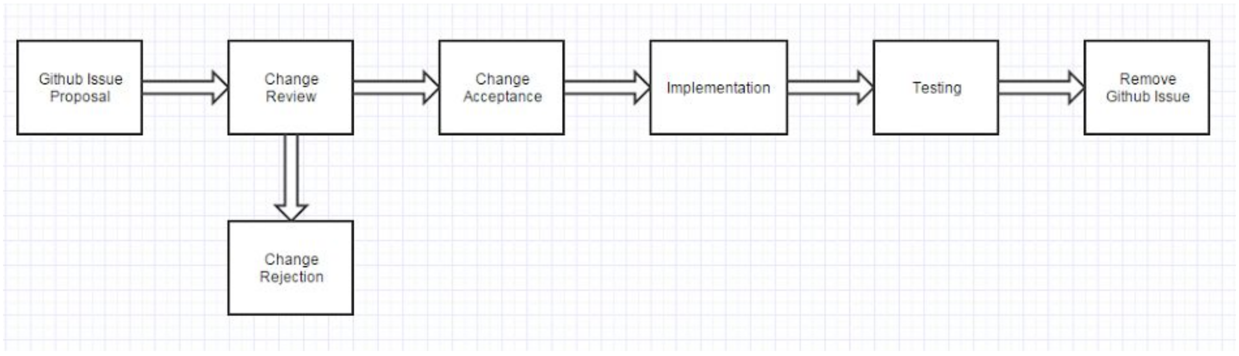


Fig 3. Representation of our Change approach. Adapted from information from [1]