[redacted]

[redacted]

Vacation Platform Development Game

[redacted]

[redacted] [redacted]

[redacted] [redacted]

[redacted] [redacted]

[redacted] [redacted]

Table of Content

1) Background

During the past, travellers could only plan their vacation through retail stores. Nowadays, vacation rental online marketplace platforms have become one of the most popular choices for people to plan their trips. Airbnb is obviously one of the biggest platforms on the Internet, which have distinct unique functions provided to the users. This platform is developed by adopting DevOps approach to continuously deploy new features and enhance services.

DevOps is one of the popular software development practices. According to Laukkarinen et al., DevOps can be defined as “practices that reduce and join repetitive tasks with automation in development, integration, and deployment” [1]. DevOps approach can help boost the speed of delivery and allow the development team continuously deploy products, as well as helping communication within the team and building solutions [2]. Continuous integration is an important development practice in DevOps which combines coding and testing by automation [3].

In this project, we are building a vacation platform development game called ‘Room HK’. The game simulates DevOps process adopted in Airbnb and focuses on Continuous integration. Players can make important incidents during the development process, and see the outcome which impacts the final result of the game. After completing the game, the players can learn DevOps concepts as well as decision-making processes during development.

2) Learning Process

During the development process of the game, our team first did research on DevOps approach adopted by Airbnb. We had read several articles and learnt DesignOps that Airbnb implemented, which design and development are combined during development, as well as the DevOps tools used for working collaboratively [4]. After that, we gathered information about unique features that are provided on Airbnb website by surfing and reading online articles. Then we designed elements included in our games, and choices that can be made by players. Finally, we breakdown the tasks and distribute them to each team member, and then hold meetings from time to time to report work progress.

3) Game Design

In this game, the main goal is to build a vacation rental online marketplace platform. Player can choose the tasks that need to be developed and assign employees to each task. There will be random incidents during the process, which need players to make decisions which have different outcomes based on the choices.

Here is the basic information and environment of the game.

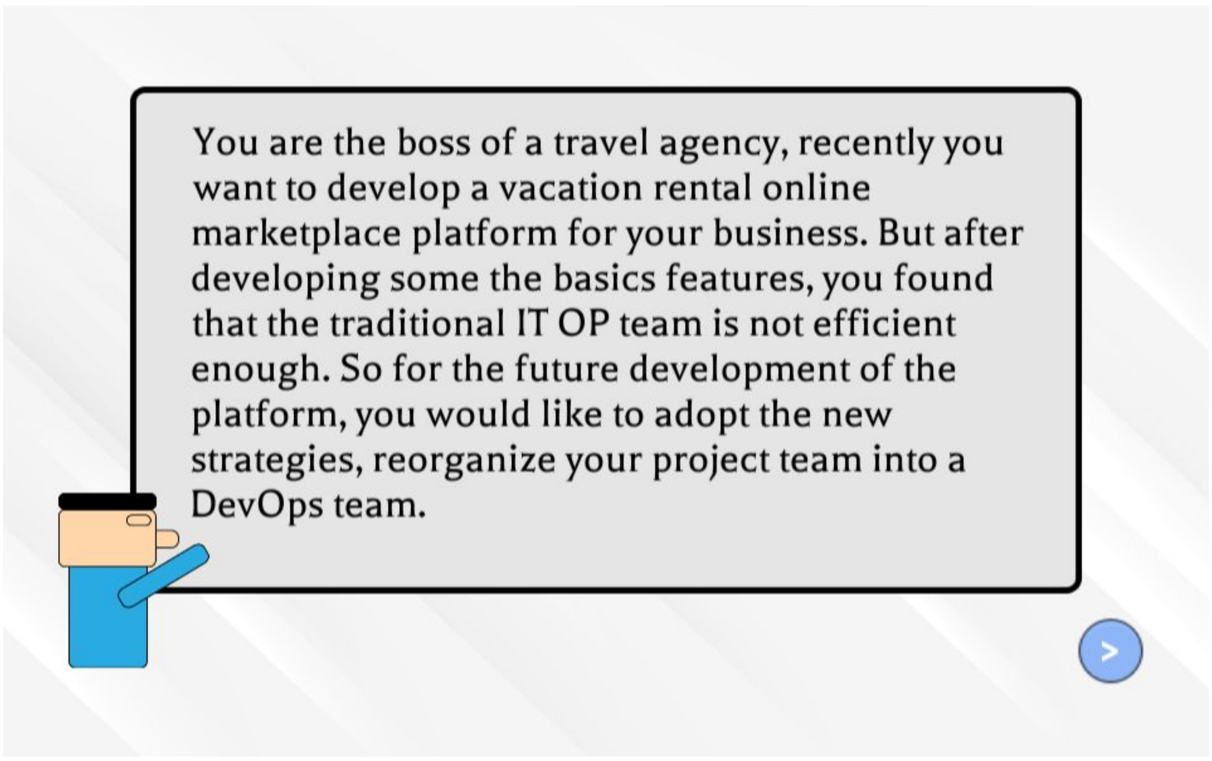
No. of Players:	1 (Single-player)
Game Time:	Around 15 minutes
Platform:	Web (HTML5 WebGL), PC (Windows, Compile Separately)
Game Engine:	Unity

[redacted]

4) Game Flow

4.1) Background Information

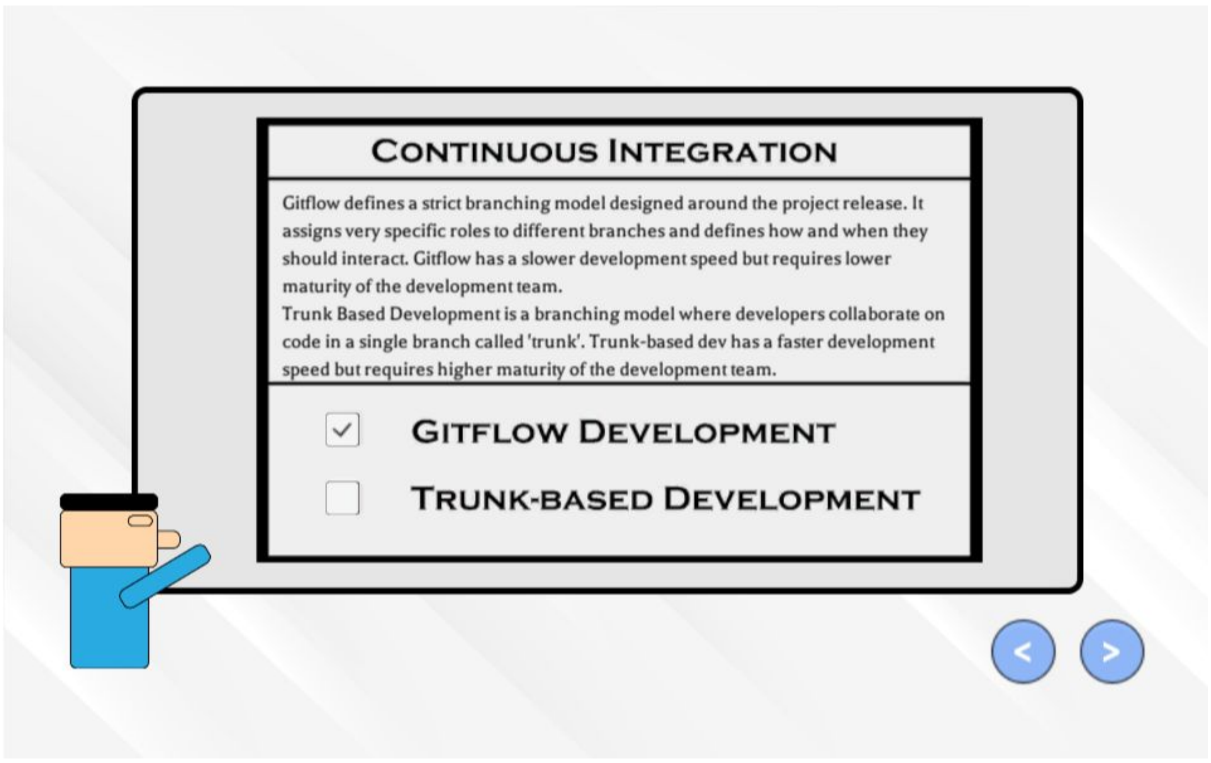
At the start of the game, the game objective and details are shown to the player. DevOps and Continuous Integration will also be introduced by providing a short description about its characteristics, so the player can first grasp the concept of the practice. Then, the game flow is introduced to the player.



You are the boss of a travel agency, recently you want to develop a vacation rental online marketplace platform for your business. But after developing some the basics features, you found that the traditional IT OP team is not efficient enough. So for the future development of the platform, you would like to adopt the new strategies, reorganize your project team into a DevOps team.

4.2) Choose a Development Practice

Before starting the development, players need to choose a development practice for continuous integration, two options are provided, the first one is GitFlow development, the second one is trunk-based development. Different source control methods have their own benefits and compromises, which could lead to different playstyles. For players choosing GitFlow, tasks could take a longer time to complete but the requirements for employees' experience is more lenient. The reason behind is that although the GitFlow practice is relatively less efficient, there are more opportunities for senior developers to supervise and review pull requests before they are merged to different branches. For Trunk Based Development, the coding and testing speed is increased but more experienced employees are referred to deliver high quality products. This is to simulate tasks reaching the master branch much quicker than GitFlow since the employees are more independent with their own commits to the trunk. However, there shouldn't be too many junior employees when adopting this workflow since it offers less opportunities to carefully review their code. For more information see part 4.4.4 (Source Control Practice Effect During Development)



CONTINUOUS INTEGRATION

Gitflow defines a strict branching model designed around the project release. It assigns very specific roles to different branches and defines how and when they should interact. Gitflow has a slower development speed but requires lower maturity of the development team.
Trunk Based Development is a branching model where developers collaborate on code in a single branch called 'trunk'. Trunk-based dev has a faster development speed but requires higher maturity of the development team.

✓ GITFLOW DEVELOPMENT

☐ TRUNK-BASED DEVELOPMENT

4.3) Choosing Employees

The player can choose the employees hired for the project. Attributes of each employee and the total cost of hiring employees will be shown. Each employee has 7 attributes, which are:

a) <u>Efficiency</u>

Efficiency determines the base coding speed of an employee. Higher efficiency indicates shorter completion time for an assigned feature. An employee having a high coding efficiency does not mean the employee can produce high quality code.

b) Coding Experience

Coding experience is a base value indicating how experienced that employee is. If the employee has more coding experience, the quality of code produced by this employee will be higher. In other words, higher coding experience represents senior developers while lower coding experience indicates junior developers.

* Coding experience is one of the two base factors increasing a product's quality. It is essential to keep products in high quality to reduce the chance of defects and downtime. (see 4.4.4, 4.4.5, 4.4.6)

c) Testing

Testing is a base value for determining the unit test writing speed of the employee as well as the quality of the tests. The quality of the test will ultimately be reflected in the quality of the product. If there is a low skill, high efficiency working for a high difficulty task. It is recommended to part with a high testing or experience employee to maintain the desired product quality.

* Testing is one of the two base factors of increasing a product's quality. It is essential to keep products in high quality to reduce the chance of defects and downtime. (see 4.4.4, 4.4.5, 4.4.6)

d) Operation / Monitor

The level of operation or monitor impacts the time required for identifying problems in production. Higher level of operation or monitor can minimize the impact of the problems and the mean time to recovery. Having employees with high operation stats could reduce the amount of downtime in production if it happens.

e) Release Engineering Skills

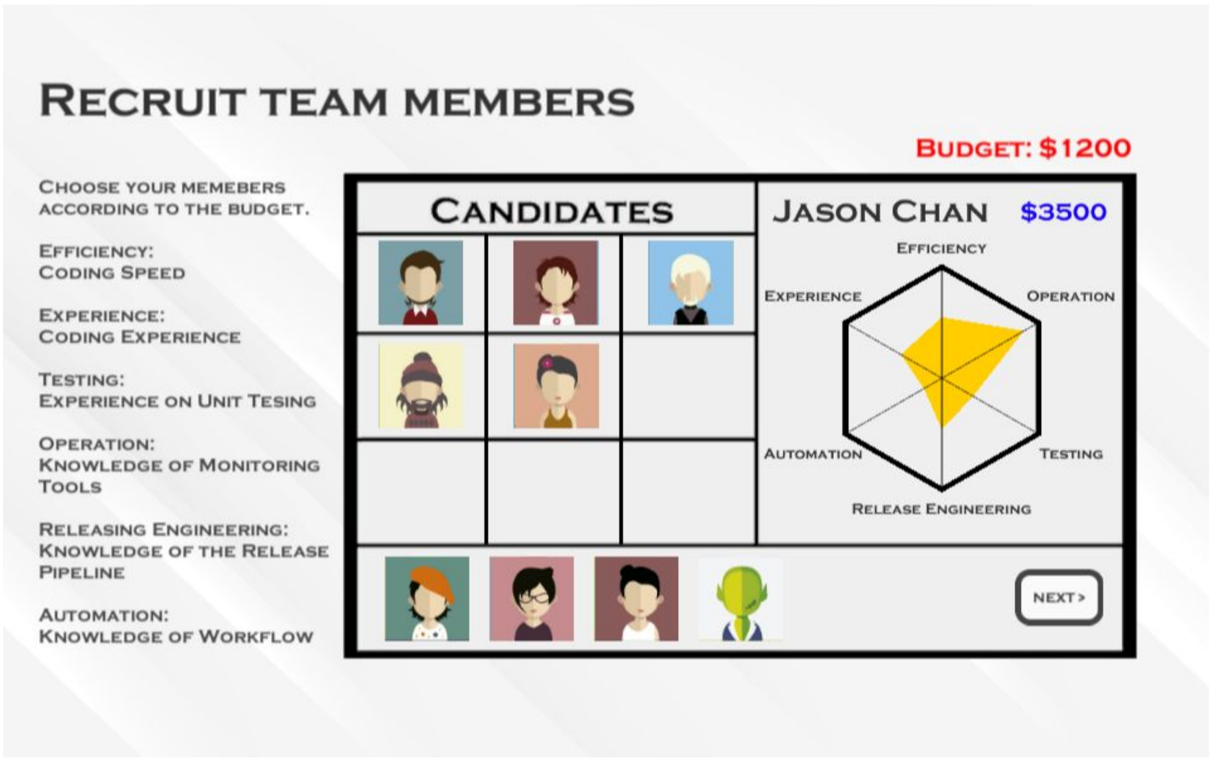
Release engineering skill affects how successful is the continuous integration process as well as automation implemented in the project. This will affect the speed of the "release stage" before the deployment of a feature. In this stage, we simulate the work being done before a release, such as cherry picking commits in the trunk-based practice or starting and finishing a release branch in the GitFlow practice.

f) Cloud and Automation Skills

The speed of implementing automation in the development depends on the cloud and automation skills of the involved employee. Higher levels of cloud and automation skills reduce the time required for unlocking and implementing certain upgrades.

g) Cost

Players need to spend money on hiring employees. Higher cost indicated an employee with higher expertise in some areas. There is a budget for the project and the total cost should not exceed the budget, so the player needs to choose wisely on which employees should be hired.


RECRUIT TEAM MEMBERS

BUDGET: $1200

CHOOSE YOUR MEMEBERS ACCORDING TO THE BUDGET.

EFFICIENCY: CODING SPEED

EXPERIENCE: CODING EXPERIENCE

TESTING: EXPERIENCE ON UNIT TESING

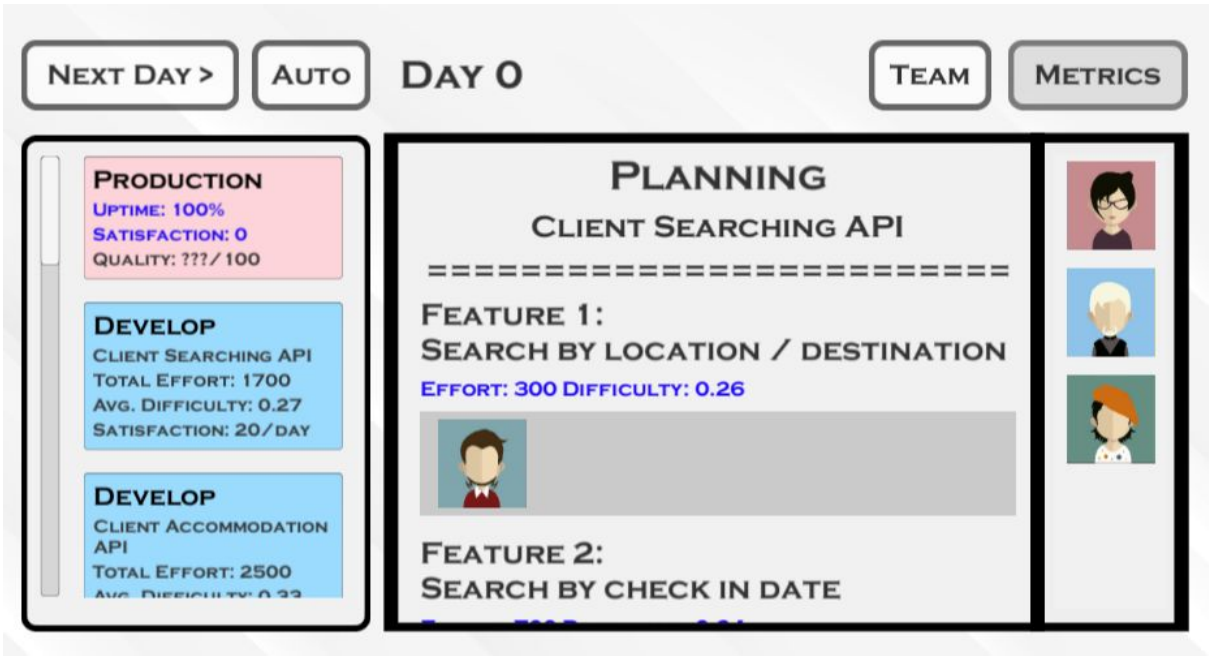
OPERATION: KNOWLEDGE OF MONITORING TOOLS

RELEASING ENGINEERING: KNOWLEDGE OF THE RELEASE PIPELINE

AUTOMATION: KNOWLEDGE OF WORKFLOW

4.4) Main Game

After hiring employees, the user will be greeted with the main game screen. Day is a basic time unit in this game (although some metrics will show hours). The time is frozen before the player clicks the “Next Day” button. Before the players end their day with the “Next Day” button, they can perform multiple tasks: choosing tasks, assigning employees to one or more tasks, view employees and team info, oversee the development process, check on production downtime, fix defects, implement upgrades and improvements, and view systems metrics. The player’s ultimate goal is to get as much satisfaction as possible as well as keeping the downtime low in production.



Production
Uptime: 100%
Satisfaction: 0
Quality: ???/100

Develop
Client Searching API
Total Effort: 1700
Avg. Difficulty: 0.27
Satisfaction: 20/day

Develop
Client Accommodation API
Total Effort: 2500

Planning
Client Searching API
========================
Feature 1:
Search by location / destination
Effort: 300 Difficulty: 0.26

Feature 2:
Search by check in date

4.4.1) Choosing Task

There are a total of 11 tasks needed to be finished during the development process. Each task has a number of features to be fully completed. There are 2 major visible attributes for each task:

a) <u>Effort</u>

Effort represents the time required to complete the features in the task. Employees with higher efficiency can finish a feature with a high effort more quickly compared to those who have lower efficiency.

b) <u>Difficulty</u>

Difficulty represents the difficulty to complete the features in the tasks. However, it does not mean that the feature will take longer to code (It depends on the effort instead). Features having a higher difficulty will have a longer testing time and higher failure rate during the build and test stage of the development. Deployment failure will also likely occur in high difficult tasks. Most importantly, the end product is also likely to be at low quality even if all stages are passed. It is highly recommended to assign employees with higher experience or testing skill to high difficulty features to negate the negative effects of a high difficulty task. Having a low quality product is very undesired as it will continuously impact the gameplay after the product is deployed.

c) Satisfaction (per day)

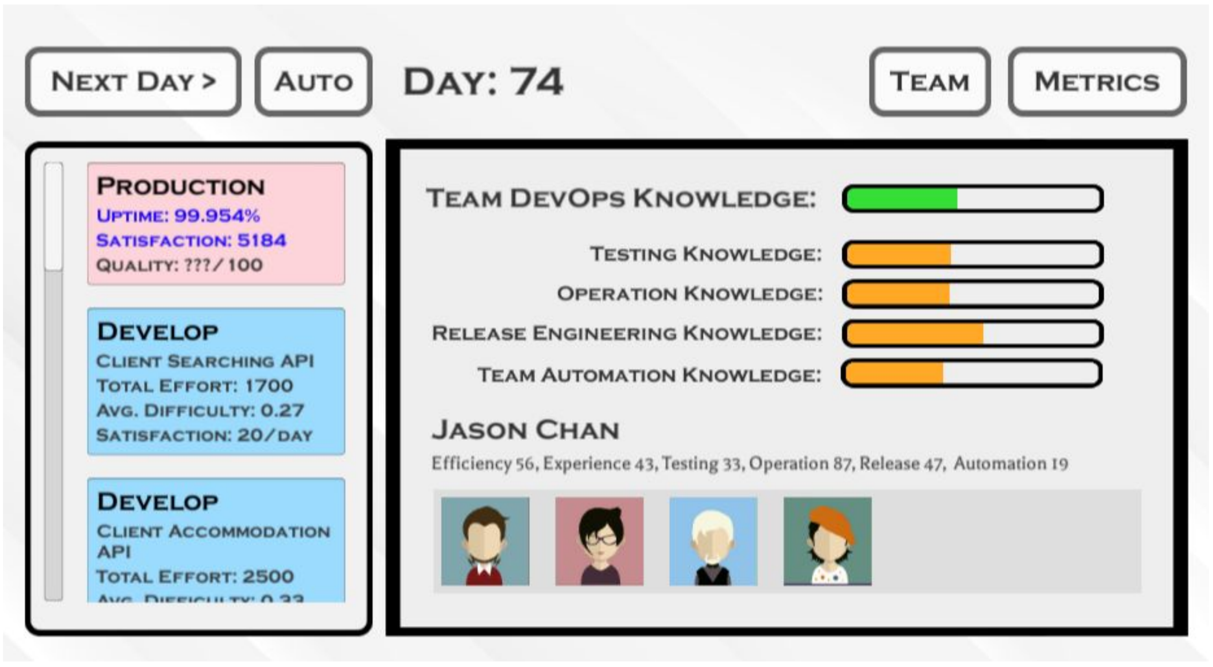
This stat is visible in the task list on the left side. It shows the base satisfaction value gained per day after the feature is deployed. However, the final satisfaction will also be affected by events after the deployment, such as the product quality and number of defects.

4.4.2) Assigning Employees

After choosing which tasks should be completed for this round, the player can then assign one or more employees to each feature of the chosen tasks. When multiple employees are assigned to the same features, their ability point stacks. That means you can have a high efficient junior coder and an experienced tester assigned to the same feature to balance it out. It is also possible to assign different tasks to the same employee and start multiple tasks at the same time. In this case, the employee's working capability is splitted to different tasks. There is a slight context-switching overhead (10%) for each of the tasks the employees are managing simultaneously. Players have to decide whether it is beneficial for them to assign multiple tasks to the same employee at a given time.

4.4.2) Checking Employees and Team Statistics

The player can check the team's statistics at any time by clicking the "Team" button above. They can also review the employee's stats by hovering over their icons in case they forget about them. Clicking the “Team” button again will retracts the team statistics and allow the player to continue what they were doing before.



Production
Uptime: 99.954%
Satisfaction: 5184
Quality: ???/ 100

Develop
Client Searching API
Total Effort: 1700
Avg. Difficulty: 0.27
Satisfaction: 20/day

Develop
Client Accommodation API
Total Effort: 2500

Team DevOps Knowledge:
Testing Knowledge:
Operation Knowledge:
Release Engineering Knowledge:
Team Automation Knowledge:

Jason Chan
Efficiency 56, Experience 43, Testing 33, Operation 87, Release 47, Automation 19

The team DevOps knowledge is a bonus that is awarded passively to the player everyday. It contains 4 categories and each category contributes 25% of the total team DevOps Knowledge. The amount of bonus awarded each day for each category depends on the aggregated statistics for all employees in the team. For example, if the team has a lot of employees with high release engineering knowledge, the player will get more team release engineering knowledge each day. The knowledge awarded each day will be higher as the team gets more knowledge in different categories. This simulates the communication between the team members and how they get used to the tools as they progress.

The player should aim to max out the Team DevOps Knowledge as soon as possible. One possible strategy is to get some knowledge in each of the categories for the team during employee selection, and not only choose employees specializing in one area. However, the player has to take other factors in the game as well (costs, employee abilities) to make their final decision.

Regardless, the player will enjoy a larger bonus in the late game which will allow them to complete more difficult tasks easier.

- Team DevOps Knowledge

  Gives a global coding speed bonus as high as +100%. This value is calculated by the sum of all sub knowledges:
    - Team Testing Knowledge (25%)
        - Gives a global testing speed bonus as high as +100%
    - Team Operation Knowledge (25%)
        - Gives downtime reduction bonus during production faults
        - Required for implementing upgrades
    - Team Release Engineering Knowledge (25%)
        - Gives a global release speed bonus as high as +100%
        - Required for implementing upgrades
    - Team Automation Knowledge (25%)
        - Required for unlocking upgrades
        - Required for implementing upgrades

4.4.4) Development Process

Once the player confirms their assignment of features, the task will be moved to the development stage. The player can continue to assign employees to other tasks or observe the progress. Clicking “Next Day” at the top left hand corner will advance the day and hence show progress. Clicking “Auto” will automatically advance the day until all employees are idle.

There are a total of 4 stages during the development process, including code, build, test and merge. After all features are merged, the release progress bar will begin to fill up on the right side of the screen to indicate the release preparation progress of the current task.

When the release bar is filled up, the product is complete. The team will then attempt to deploy the features. Depending on the quality of the product, the deployment might fail, which could further delay the progress. Once the features are deployed, the task is moved to the deployed state.

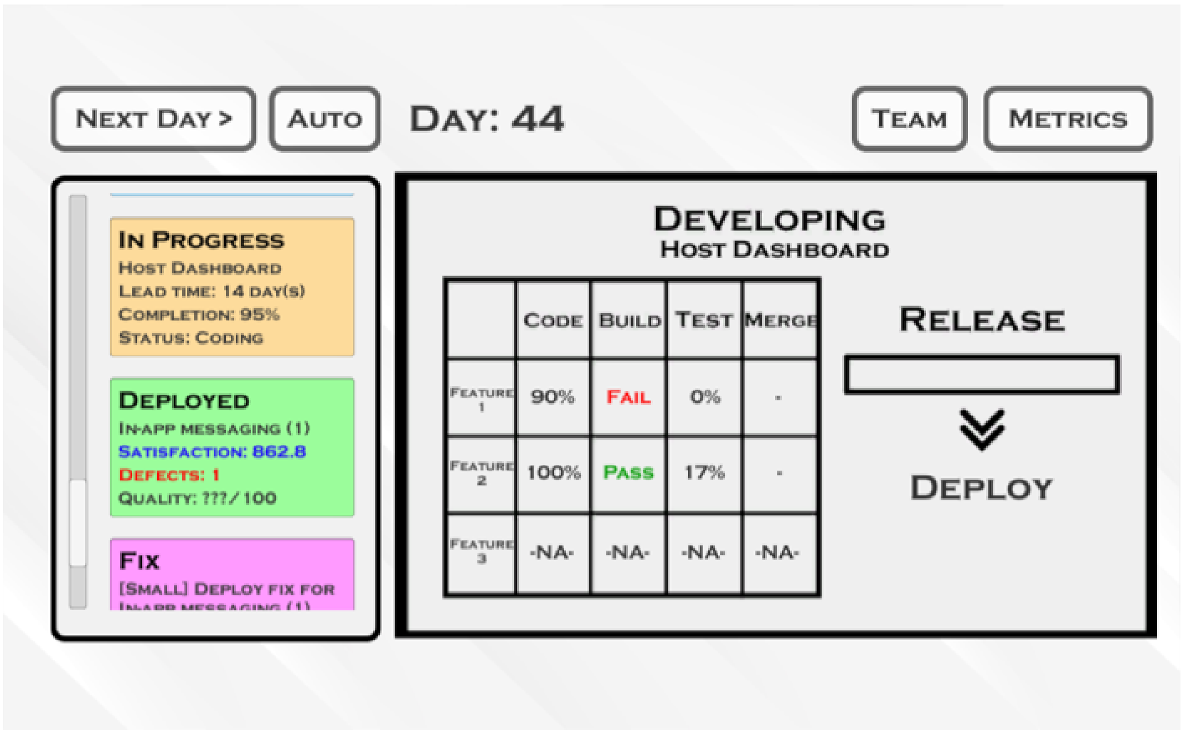
Product Quality

Product quality is determined during the development process. Lots of factors contribute to the quality of the product: the difficulty of the feature, the aggregated experiences of the assigned employees and the aggregated testing ability point of the assigned employees. Basically, if employees with low experience and testing skills are assigned to a high difficulty feature, the overall product quality will suffer severely.

Product quality is a statistics originally hidden from the user. However, this value is closely related to the probabilities of build failure, test failure, deployment failure and defects. The player is able to know from the statistics and metrics that a deployed task is suffering from low product quality.

Low quality products are more harmful after the deployment, as defects will cause the performance of the features and even the overall production environment to degrade, reducing the satisfaction gained each day as well as increasing the likelihood of a production downtime.

- Build failure can occur when the code progress fills up to 100%. In this case, the code progress will roll back a certain amount for bug fixing.



DAY: 44

IN PROGRESS
HOST DASHBOARD
LEAD TIME: 14 DAY(S)
COMPLETION: 95%
STATUS: CODING

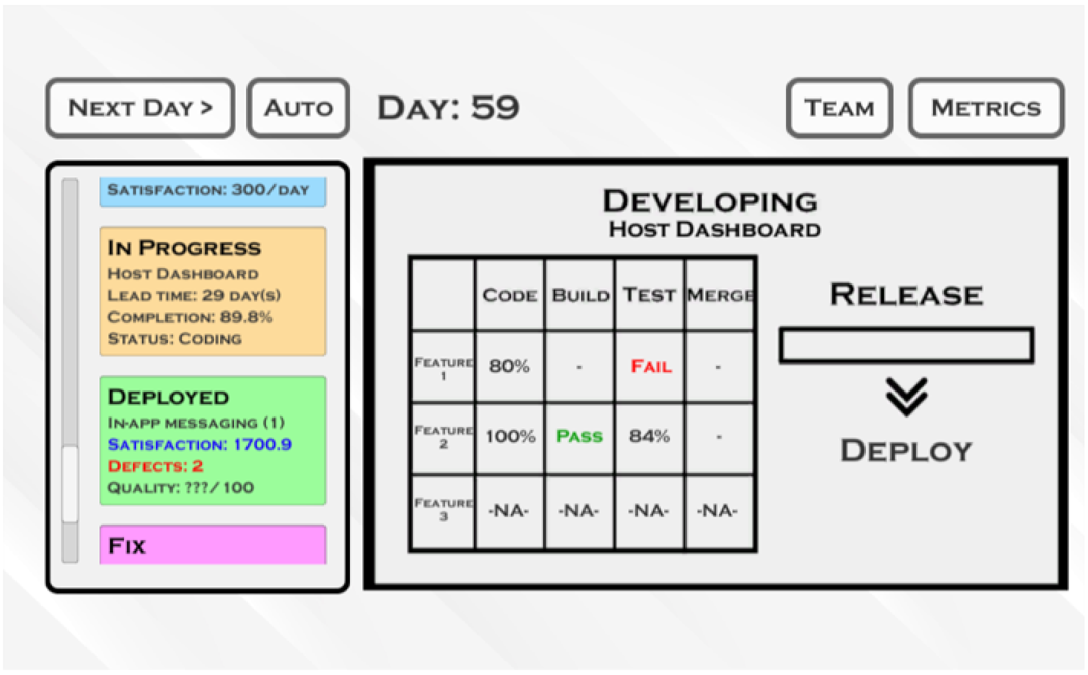
DEPLOYED
IN-APP MESSAGING (1)
SATISFACTION: 862.8
DEFECTS: 1
QUALITY: ???/100

FIX
[SMALL] DEPLOY FIX FOR

DEVELOPING
HOST DASHBOARD

	CODE	BUILD	TEST	MERGE
FEATURE 1	90%	FAIL	0%	-
FEATURE 2	100%	PASS	17%	-
FEATURE 3	-NA-	-NA-	-NA-	-NA-

- Test failure can occur when the test progress fills up to 100%. In this case, the code progress will roll back to coding for bug fixing. However, since the unit tests are almost complete in this stage, there is very less penalty of the test progress so the developers do not have to go through the unit test writing stage all over again.



DAY: 59

SATISFACTION: 300/DAY

IN PROGRESS
HOST DASHBOARD
LEAD TIME: 29 DAY(S)
COMPLETION: 89.8%
STATUS: CODING

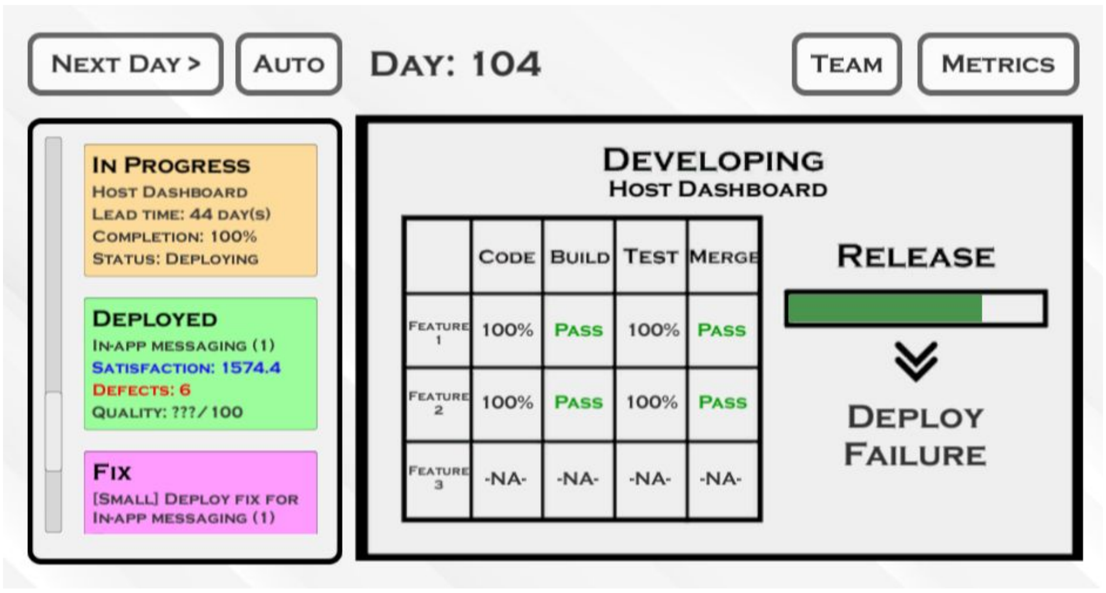
DEPLOYED
IN-APP MESSAGING (1)
SATISFACTION: 1700.9
DEFECTS: 2
QUALITY: ???/100

FIX

DEVELOPING
HOST DASHBOARD

	CODE	BUILD	TEST	MERGE
FEATURE 1	80%	-	FAIL	-
FEATURE 2	100%	PASS	84%	-
FEATURE 3	-NA-	-NA-	-NA-	-NA-

- Deploy failure happens rarely for high quality products but very frequently for unacceptably low quality products. If the deployment fails, additional time will be needed to fix the bugs and prepare the release again.



In Progress
Host Dashboard
Lead time: 44 day(s)
Completion: 100%
Status: Deploying

Deployed
In-App Messaging (1)
Satisfaction: 1574.4
Defects: 6
Quality: ???/100

Fix
[Small] Deploy fix for In-App Messaging (1)

Developing
Host Dashboard

	Code	Build	Test	Merge
Feature 1	100%	Pass	100%	Pass
Feature 2	100%	Pass	100%	Pass
Feature 3	-NA-	-NA-	-NA-	-NA-

<u>Source Control Practice Effect During Development</u>

All constants in the game are defined in the source code *Constants.cs* to allow easy game balance tweaks. Different source control practices will have different sets of constants. Here are some examples of difference.

Gitflow beneifts:

- +10% base build success probability than Trunk Based
- +10% base test pass probability than Trunk Based
- ~10% reduction of maximum experience point requirement than Trunk Based
    - *Requires less experience from employees for difficult features
- ~15% reduction of maximum testing point requirement than Trunk Based
    - *Requires less testing skills from employees for difficult features

Trunk Based benefits:

- +50 Overall coding effort factor to all employees than GitFlow (11% bonus)
    - Each employee will be able to complete 11% more coding effort each day

- +80 Overall testing effort factor to all employees than GitFlow (67% bonus)
    - Each employee will be able to complete 67% more testing effort each day

Hence, players choosing GitFlow will have to spend notably more time on testing than players who choose the Trunk Based method. However, they get slightly higher quality products in return without too much dependence on high experience or high testing skill employees.

Players choosing the Trunk Based method will find it easy to complete tasks in a notably shorter lead time. Having a larger work capacity for each employee each day, Trunk Based players might find assigning multiple tasks to the same employee a more viable strategy early game than GitFlow players as the tasks are no longer easily stuck at the testing phases for a long time.

Trunk Based players will start gaining satisfaction early in the game than GitFlow players. However, the quality of the products might suffer if not enough skilled employees or testers are assigned to the high difficulty features (which sometimes might be unavoidable). Trunk based players should focus on completing low difficult tasks as soon as possible and avoid working on high difficulty tasks until ways of improving the products' quality is available (see 4.4.7 Upgrades). Otherwise, the defects might soon put the player in disadvantage than GitFlow players.

Players should experiment with the two practices to determine which one suits their playstyles. There is no absolute correct answer.

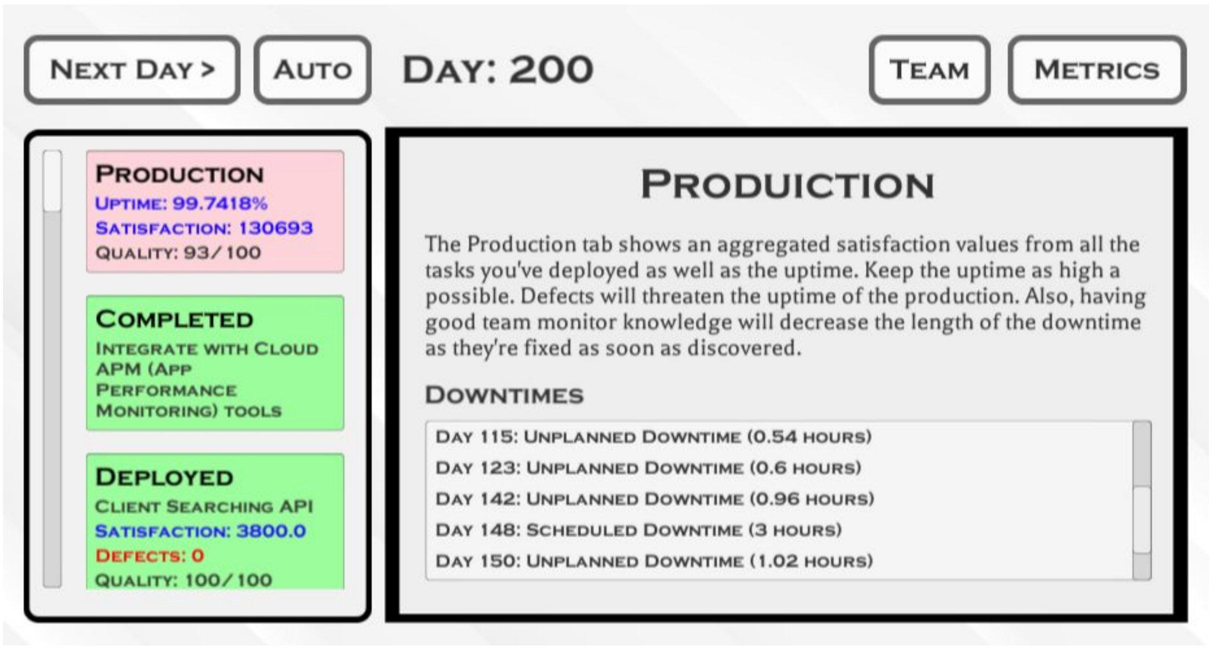
Note that the values might be changed in later patches if game balancing is necessary to make sure one source control practice is not too "overpowered" than the other.

4.4.5) Deployed Tasks and Production

Deployed tasks will show green in the task list to the left. Satisfaction shows the total satisfaction gained for the task, which will accumulate by the defined base satisfaction value (see 4.4.1) each day. If the task is deployed with quality < 100%, the satisfaction gained will be reduced to *base satisfaction * quality*. The number of defects are shown in red for each deployed task. Having defects will heavily reduce the satisfaction value gained per day for the task (8% per defect).

Having defects in any of the deployed tasks will also negatively affect the production quality (0.5% per defect, with a hard cap at 15%).

Clicking on the “Production” task in the task list will bring up a list of downtime that happened to the production environment. Some downtimes are unplanned and some are scheduled (see 4.4.6 Defects). The “Uptime” value under the production tasks shows the current uptime calculated from the downtimes. The “Satisfaction” value under the production tasks shows the total satisfaction value from all the deployed tasks. The current production environment also generates a tiny amount of satisfaction by itself (80 * Production Quality). Note that the “Quality” statistics might not be visible before upgrades are implemented (see 4.4.7 Upgrades).



PRODUCTION
UPTIME: 99.7418%
SATISFACTION: 130693
QUALITY: 93/100

COMPLETED
INTEGRATE WITH CLOUD APM (APP PERFORMANCE MONITORING) TOOLS

DEPLOYED
CLIENT SEARCHING API
SATISFACTION: 3800.0
DEFECTS: 0
QUALITY: 100/100

PRODUICTION

The Production tab shows an aggregated satisfaction values from all the tasks you've deployed as well as the uptime. Keep the uptime as high a possible. Defects will threaten the uptime of the production. Also, having good team monitor knowledge will decrease the length of the downtime as they're fixed as soon as discovered.

DOWNTIMES

DAY 115: UNPLANNED DOWNTIME (0.54 HOURS)
DAY 123: UNPLANNED DOWNTIME (0.6 HOURS)
DAY 142: UNPLANNED DOWNTIME (0.96 HOURS)
DAY 148: SCHEDULED DOWNTIME (3 HOURS)
DAY 150: UNPLANNED DOWNTIME (1.02 HOURS)

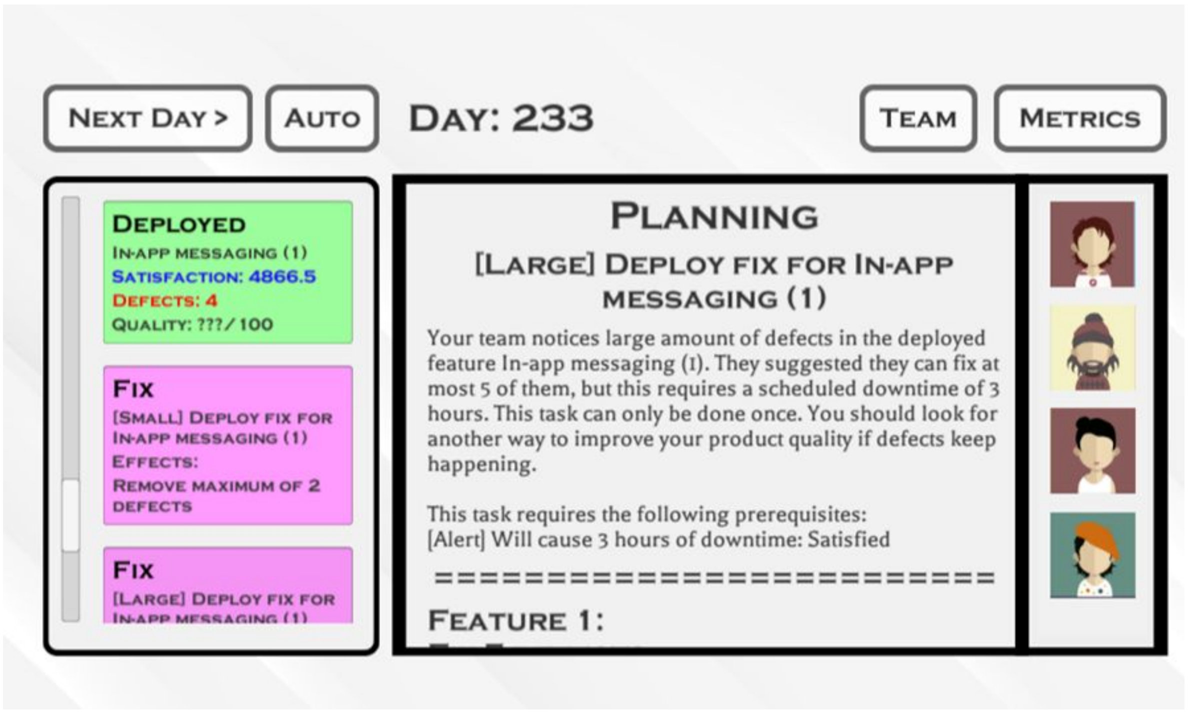
4.4.6) Defects

After the tasks have been deployed, the players might discover defects in the deployed tasks. The number of defects is shown in the task list on the left side. Defects happen randomly and their chance of happening depends on the product quality. Having high quality deliverables can drastically reduce the chance of getting a deflect.

Players could choose to assign employees to solve the defects or ignore them when developing other tasks. There would be two options for players, the first one to solve a few defects in a small scale (maximum 2 defects, visible when defect count > 1), the second one is to temporarily shut down the server and solve a large number of defects (maximum 5 defects, visible when defect count > 3). The second option will introduce a 3 hours of downtime to the production environment.

Players should make a decision to strike a balance between system downtime and the product quality (number of defects).

Note that solving defects after the tasks are deployed will **not** necessarily increase the product quality. The player should find other ways to improve the product quality as soon as possible (improvement upgrades, see 4.4.7), or develop high quality products in the first place (by assigning senior coders and testers to difficult features). Otherwise, defects could still accumulate overtime, and the defects might not be able to get fixed any further.



DEPLOYED
IN-APP MESSAGING (1)
SATISFACTION: 4866.5
DEFECTS: 4
QUALITY: ???/100

FIX
[SMALL] DEPLOY FIX FOR IN-APP MESSAGING (1)
EFFECTS:
REMOVE MAXIMUM OF 2 DEFECTS

FIX
[LARGE] DEPLOY FIX FOR IN-APP MESSAGING (1)

PLANNING
[LARGE] DEPLOY FIX FOR IN-APP MESSAGING (1)

Your team notices large amount of defects in the deployed feature In-app messaging (1). They suggested they can fix at most 5 of them, but this requires a scheduled downtime of 3 hours. This task can only be done once. You should look for another way to improve your product quality if defects keep happening.

This task requires the following prerequisites:
[Alert] Will cause 3 hours of downtime: Satisfied

========================

FEATURE 1:

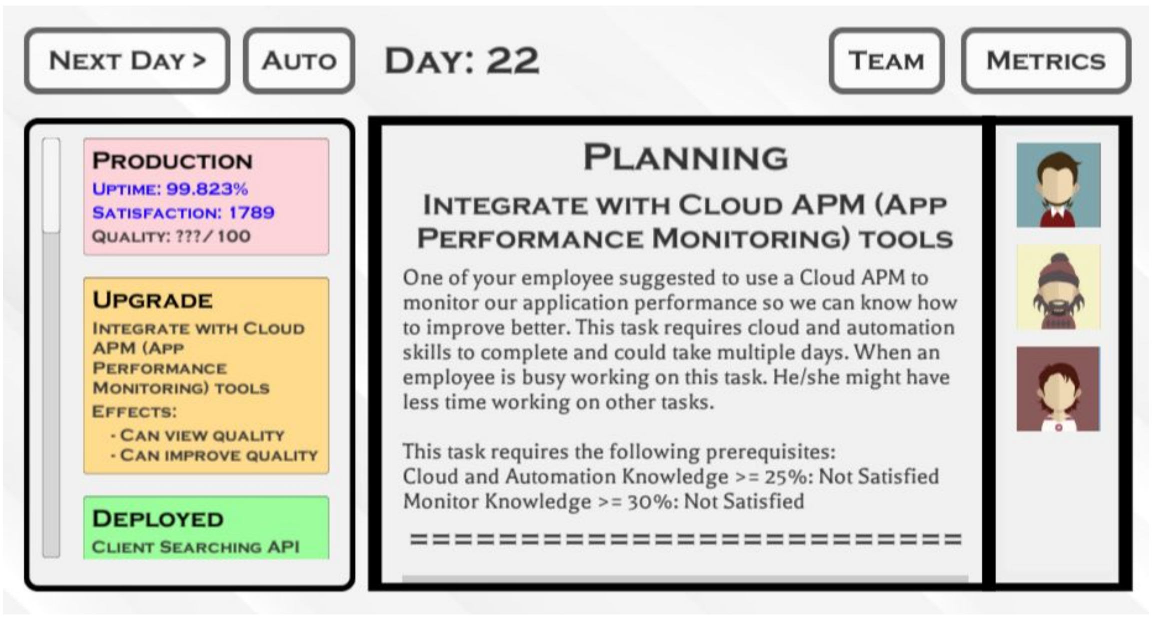
4.4.7) Upgrades

In order to smoothen and accelerate the development process of DevOps, different upgrades would be provided in the task lists for players to implement, the upgrades are basically different DevOps tools including but not only the application monitoring tools, automation tool like Jenkins, container platform like Docker.

Players could optionally choose to implement these tools based on their current team knowledge and the product quality etc., and different upgrades may cause different effects or unlock new features in the game.

Upgrades are usually not visible at the start of the game. They have some requirements to meet before they are shown to reduce clutter in the task list. Some upgrades require some prerequisites to be able to start even if they are visible to the player. The requirement will be listed in the description when the player clicks on it.

For example, starting from day 10, if the player has successfully deployed a feature, an upgrade about application monitoring tools will be shown. If requirements are met, the players can choose to adopt a cloud application monitoring solution so they could monitor the product quality while developing the tasks. If the player chooses to implement the monitor tool, the team will be able to assess the quality of the deployed products. The product quality field will be revealed to the player. Improvement tasks are unlocked so the player can increase the product quality by a maximum of 20%.



Day: 22

Production
Uptime: 99.823%
Satisfaction: 1789
Quality: ???/100

Upgrade
Integrate with Cloud APM (App Performance Monitoring) tools
Effects:
- Can view quality
- Can improve quality

Deployed
Client Searching API

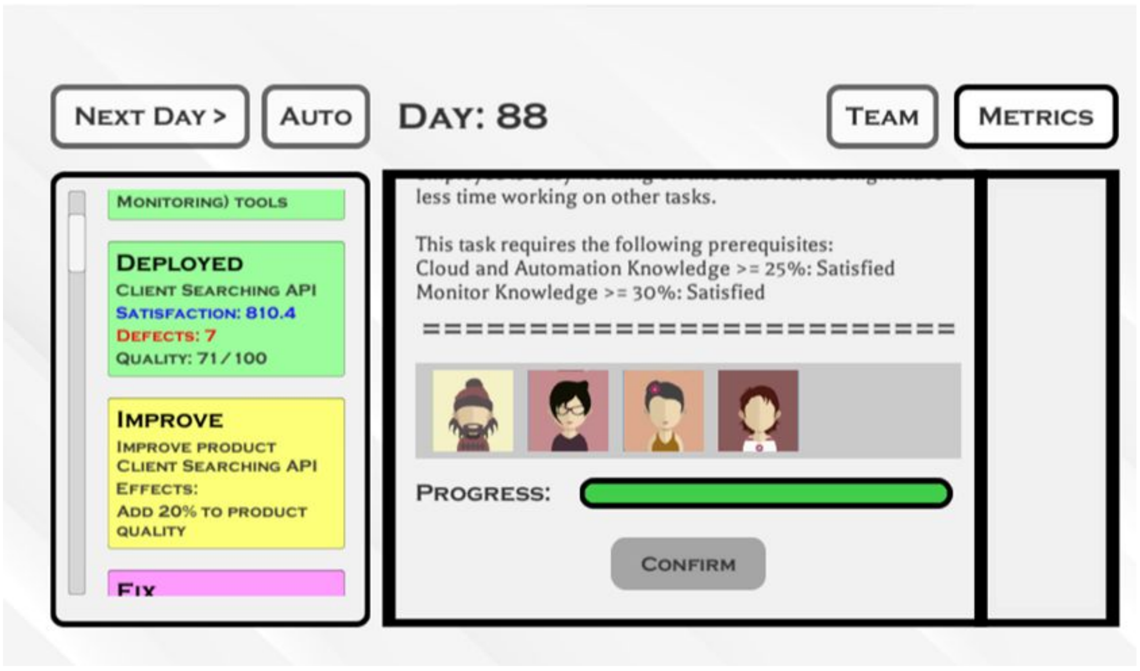
Planning

Integrate with Cloud APM (App Performance Monitoring) tools

One of your employee suggested to use a Cloud APM to monitor our application performance so we can know how to improve better. This task requires cloud and automation skills to complete and could take multiple days. When an employee is busy working on this task. He/she might have less time working on other tasks.

This task requires the following prerequisites:
Cloud and Automation Knowledge >= 25%: Not Satisfied
Monitor Knowledge >= 30%: Not Satisfied

(This task requires Team Automation Knowledge > 25% and Monitor Knowledge > 30%)

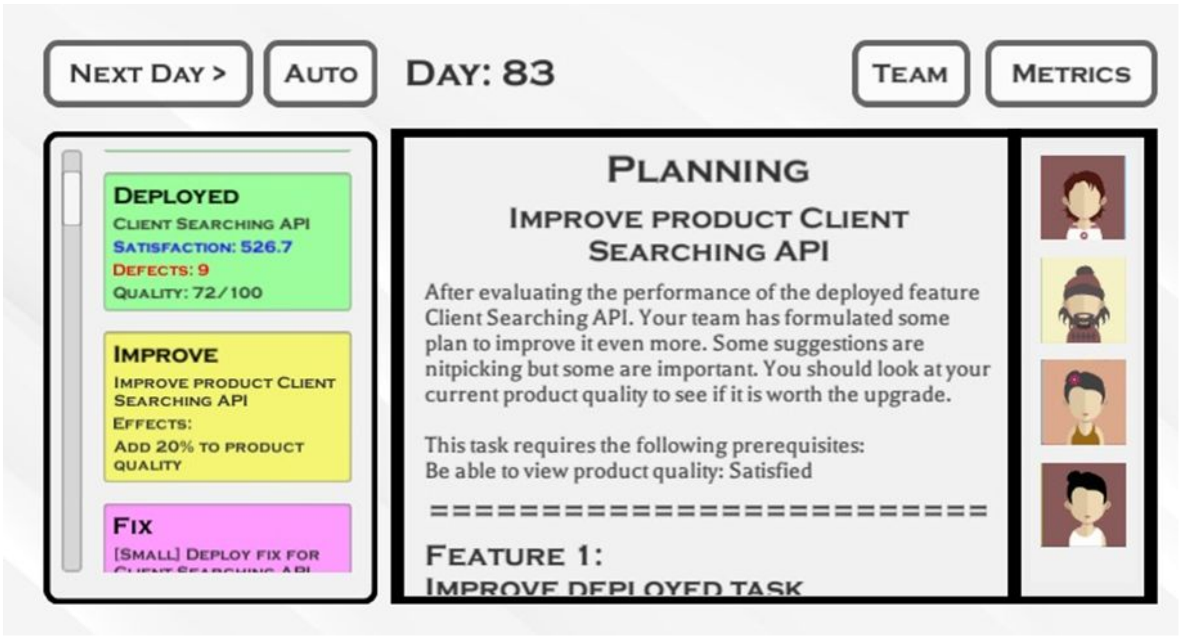


less time working on other tasks.

This task requires the following prerequisites:
Cloud and Automation Knowledge >= 25%: Satisfied
Monitor Knowledge >= 30%: Satisfied

==========================

(Improvement tasks are unlocked under every deployed tasks that is not 100% quality)



Planning

Improve product Client Searching API

After evaluating the performance of the deployed feature Client Searching API. Your team has formulated some plan to improve it even more. Some suggestions are nitpicking but some are important. You should look at your current product quality to see if it is worth the upgrade.

This task requires the following prerequisites:
Be able to view product quality: Satisfied

==========================

Feature 1:

Improve deployed task

(An example of the improvement tasks)

<u>Extensibility</u>

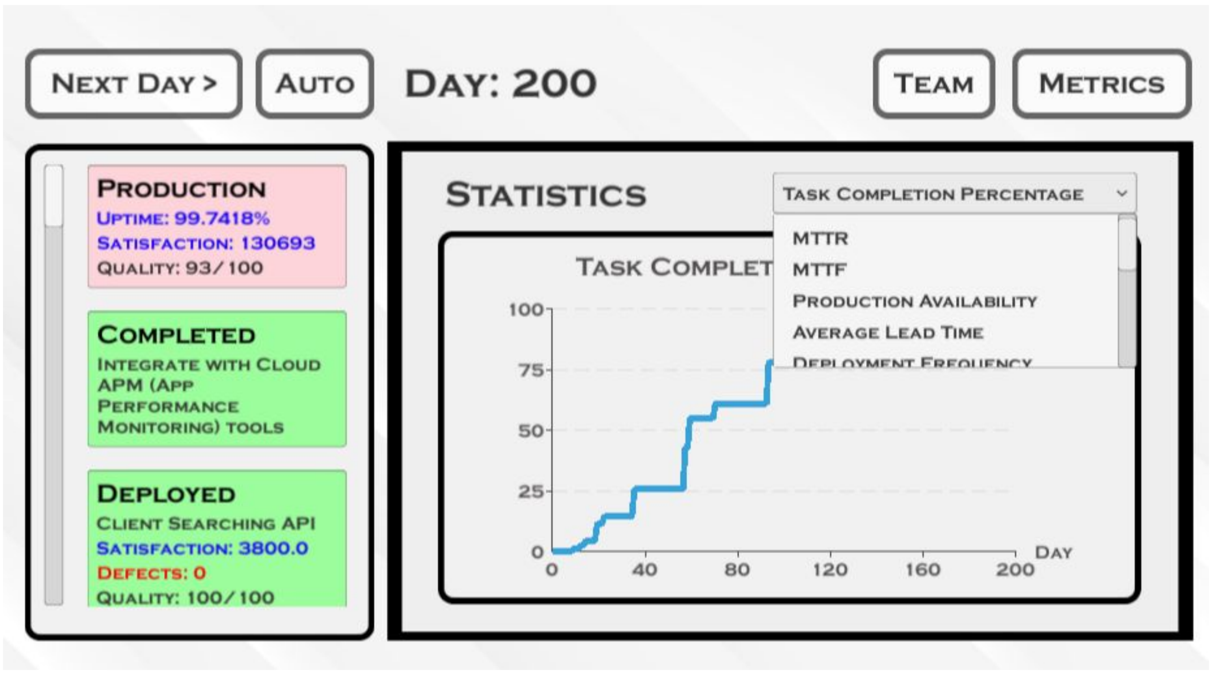
Currently, there are only very few upgrade tasks due to time limitations. However, the *SimpleTask.cs* class is defined in the source code to allow more tasks to be added to the game in a few lines of code. With the *SimpleTask.CreateTask*()* method, one can define different types of tasks without creating a new class or modify other existing game code. The custom *SimpleTask* can have different names, descriptions, features to implement, visible requirements, unlock requirements, task finish effects and even custom employee work logic (such as using other employee abilities to finish a task). It can access the game state, metric and statistics provided by the *GameManager*. Once the tasks are created, one can then add them inside *TaskFactory.cs*. Refer to *SimpleTask.cs* and *TaskFactory.cs* for more information. In the future, more tasks might be added with the SimpleTask API.

4.5) System Metrics

After the development process, system metrics are shown based on the player's decisions and development progress. A trend graph is shown to represent system metrics for each day, as well as other statistics, including task completion percentage, mean time to recover, mean time between failures, production availability, average lead time, deployment frequency, change fail percentage, frequency of production failure, build failure count, test failure count, test user satisfaction. Hovering over the chart can view the exact values. Player can measure their performance by investigating those statistics.

- Task Completion Percentage
  - Total Completed Effort / Total Effort of All Tasks
- Mean Time To Recover (in hours)
  - The MTTR values calculated from the downtime reports
- Mean Time To Failure (in hours)
  - The MTTF values calculated from the downtime reports
- Production Availability
  - The production availability
- Average Lead Time
  - Average lead time for all deployable tasks (tasks with satisfaction values)
- Deployment Frequency
  - Deployment frequency for all deployable tasks (tasks with satisfaction values)
- Change Fail Percentage
  - Percentage of deployment that is failed

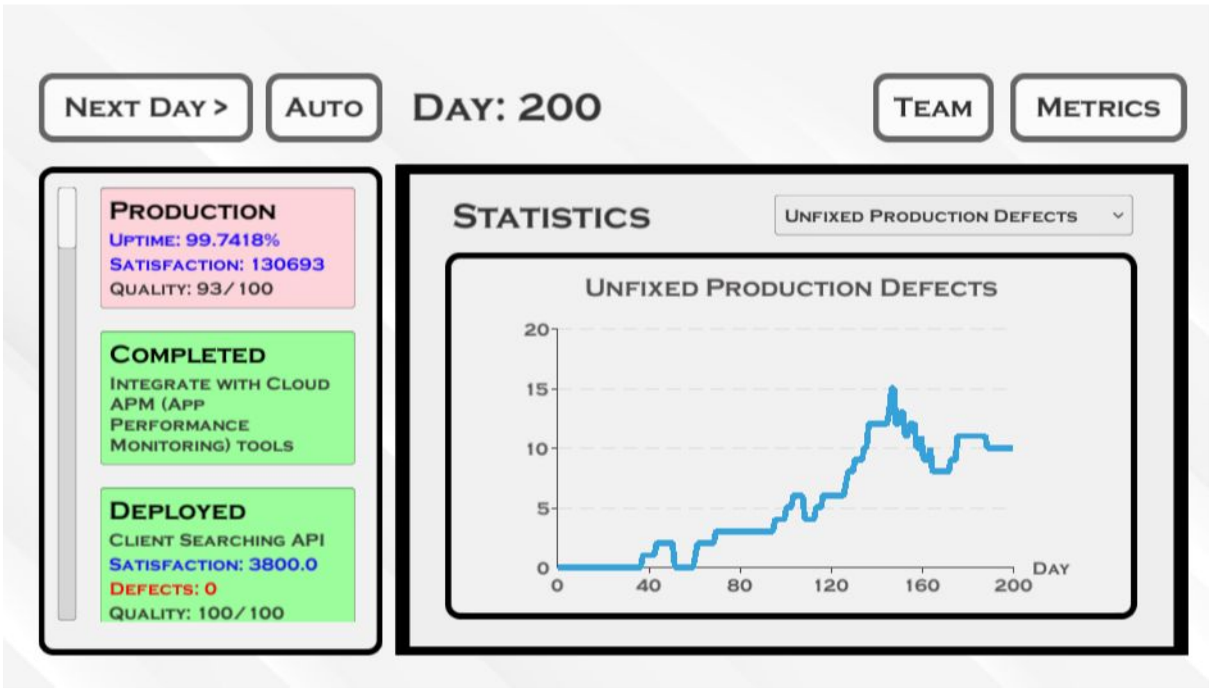
- User Satisfaction
  - Total user satisfaction (production + all deployed tasks)
- Build Failure Count
  - Total number of build failure count from all tasks
- Test Failure Count
  - Total number of test failure count from all tasks
- Build Failure Percentage
  - number of build failure / total builds from all tasks
- Test Failure Percentage
  - number of test failure / total tests from all tasks
- Total Production Defects
  - An accumulated graph of all discovered defects
- Unfixed Production Defects
  - Number of unfixed defects
- Defect Frequency
  - Average number of defects per day
- * Knowledge
  - All DevOps knowledges graph



Production
Uptime: 99.7418%
Satisfaction: 130693
Quality: 93/100

Completed
Integrate with Cloud APM (App Performance Monitoring) tools

Deployed
Client Searching API
Satisfaction: 3800.0
Defects: 0
Quality: 100/100



Production
Uptime: 99.7418%
Satisfaction: 130693
Quality: 93/100

Completed
Integrate with Cloud APM (App Performance Monitoring) tools

Deployed
Client Searching API
Satisfaction: 3800.0
Defects: 0
Quality: 100/100

4.5.1) Score

Score is an unimplemented feature of the game originally planned to show when the game has reached a determined amount of days. The score should be calculated from the system metrics collected from the game. The final score will be shown to the player to reflect on their performance throughout the game. The score reflects availability, reliability and scalability. These results are calculated based on the player's decisions in the development process, the two major variables that would affect the score are the customer satisfaction and the system uptime.

a) Availability

Availability indicates the proportion of time that the system is functional and working. The higher availability is, the longer the platform is operational. The player should reach a defined minimum availability requirement otherwise a huge penalty will be applied to the final score.

b) Reliability

Reliability indicates the duration of platform functions probably. Higher level of reliability indicates the platform can produce correct output in a longer period of time, it also reflects the efficiency of handling and solving sudden accidents during the development process.

c) <u>Scalability</u>

Scalability depends on the changes on project scale, including both expanding and scaling down. This attribute is affected by collaboration within the development team, automation during the development process, investment in deployment and monitoring and risk during the release stage.

5) Game Strategies

Here are some strategies for playing the game.

- Consider the skill of each employee seriously, and only hire new employees who have unique skills compared to those hired so as to spend the money wisely.
- Pay attention to the completion progress of each task and consider whether to start a new task or finish the incomplete tasks first.
- Remember the concepts of DevOps and how it can benefit the development process to make decisions wisely.
- System metrics and feedback can be a great help on decision-making processes and help the player reconsider the strategy of assigning employees to different features.
- Remember to read user satisfaction so as to improve the platform and change those aspects which do not meet users' expectations.

6) Game Decisions and Dilemmas

Major decisions and dilemmas in the game includes:

- Choose between the two development practices
- Choose employees with balanced skills but within budget

- Number of tasks assigned to an employee concurrently
- Number of employees assigned to a feature
- When and what upgrades to be implemented
- When and how to solve the defects

Appendix: Work Breakdown

[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

Reference

[1] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, *“Regulated software meets DevOps,”* Information and Software Technology, vol. 97, pp. 176–178, 2018.

[2] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, *“DevOps,”* IEEE Software, vol. 33, no. 3, pp. 94–100, 2016.

[3] C. Amrit and Y. Meijberg, *“Effectiveness of TestDriven Development and Continuous Integration,”* IT professional, vol. 20, no. 1, pp. 27–35, 2018.

[4] E. Dornenburg, *“The Path to DevOps,”* IEEE Software, vol. 35, no. 5, pp. 71–75, 2018.