CMPE/CISC 326
A3: Enhancement Proposal
SuperTux

Group: TBA

December 5th, 2017

Instructor: Ahmed Hassan

Coco Chen 14kxc@queensu.ca
Yihao Chen 15yc9@queensu.ca
Yuhao Chen 14yc37@queensu.ca
Brayden Dewar 13bad2@queensu.ca
Lena Krause 14lk6@queensu.ca
Selin Onsoz 13bso@queensu.ca

Abstract

The purpose of this report is to introduce a feature to add to the open source side scrolling video game SuperTux. The proposed feature is a shop that allows the player to exchange in-game coins for certain power ups. The implementation of the feature requires modifications to several subsystems. The impact of these modifications will be investigated with respect to the conceptual architecture and concrete architecture of the system.

The shop allows players to safely acquire power ups in exchange for coins. The motivation for this feature is to give the user the choice to lower the difficulty of a level by taking advantage of the shop. The inside of the shop was decided to its own 'level'. The user enters the shop and controls Tux as normal. Several blocks are displayed, each one containing a distinct item that the player can activate, dispensing the item and removing coins from the inventory. The player can exit the shop to the right of the screen. Two methods were proposed for integrating the shop into the game – The shop is a separate level the player enters via the world map, or the shop is automatically entered on the initialization of every level. A SAAM analysis was used to conclude that integrating the shop as a separate level on the world map is the more effective option because of the increased user satisfaction and the ease of implementation.

The feature was determined to have dependencies with the Audio, Utilities, and Game Data subsystems. The feature required modifications to the User Input, Game State, User Interface, and Graphics subsystems. Testing of the feature was planned to have two phases – testing as a standalone feature, and testing it as implemented in the game. The aspects requiring most attention was creating the blocks that deduct coins when activated and imposing the restrictions on circumstances for activation of these blocks.

The feature was not determined to have an impact of the concurrency of the system. SuperTux was observed to handle concurrency via a thread queue. Team issues include the difficulty of retrieving source code for files created in the level editor and the lack of documentation in the source code. Lessons learned by the team were the importance of a SAAM analysis to compare methods with regard to stakeholder needs & wants, and non-functional requirements.

Contents

Abstract **1**

Contents **2**

1.0 Introduction **3**

2.0 Proposal Description **3**

2.1 Approaches 4

3.0 SAAM Analysis **5**

3.1 User Satisfaction 6

3.2 Simplicity, Testability, Maintainability 7

3.3 Extensibility 7

3.4 Performance 7

3.5 Chosen Implementation 7

4.0 Impact on Architecture **8**

Impact of Dependency 8

Changing 9

5.0 Sequence Diagrams **9**

6.0 Risks & Limitations **10**

7.0 Testing **10**

8.0 Concurrency and Team Issues **11**

Concurrency 11

Team Issues 11

9.0 Lessons Learned **11**

10.0 Conclusion **12**

11.0 References **13**

Implementation Description 13

1.0 Introduction

Project 1 & 2 focused on investigating the architecture of SuperTux. The subsystems were defined as well as the data flow and connections between subsystems. The architectural style can be classified as a primarily layered system. The higher level subsystems in the conceptual architecture are the User Interface, Game Specific Subsystems, and Resources. This project expands on the prior by investigating how to implement a shop feature into SuperTux's architecture.

The conclusion is that implementing a shop as a separate level accessed through a portal on the world map is the optimal method for maximizing stakeholder wants and needs. No new subsystems are required for this feature, however some subsystems were modified. The impact architecture of the feature allows the team to view how the feature affects the subsystems in the game.

The proposed feature was implemented successfully and tested rigorously. Details of the implantation are provided at the end of this report.

2.0 Proposal Description

Currently, SuperTux does not have any shop feature. Shops have become a commonplace in modern video games and are also present in the Super Mario games which SuperTux draws heavy influence from. The coins in SuperTux collected in the game are solely used for score purposes. This feature adds the additional functionality of coins to be used as currency in exchange for power ups. The goal of the shop is to allow this exchange to take place in an area where the user is safe from enemy attacks. The primary motivation and effect that this feature has on the user experience is to give the user the option to decrease the difficulty of a level by taking advantage of the power ups offered in the shop.

The shop is to have several common power ups ranging from the egg which allows tux to grow and the flowers that give Tux special abilities. The shop will be created using the level editor. Players can purchase items by activating special blocks that both dispense a power up and deduct coins from the player inventory. These blocks are to be custom made by making modification to Game Object files. The user can exit the shop with or without purchasing an item by taking an exit on the right of the screen.

Fig 1: Concept image of interior of shop

2.1 Approaches

Approach 1: The shop is a separate level in the world map

In this approach the user enters the shop through a portal on the world map similar to normal levels. The player is brought to the shop level where they conduct their business. After exiting the shop, the user will be brought back to the shop portal on the world map. This approach allows the user to enter the shop only when they want to. The user can also ignore the shop completely if they feel that they have no use for it. This approach would require every world to be customized to accommodate the shop portal, but this can be done with the level editor rather than through modifying source code

Approach 2: The shop is initiated before every level

In this approach, every time the user initiates a level on the world map, they will be brought to the shop. The player can conduct their business in the shop, or quickly exit the shop by running to the exit. After exiting the shop, the player will be brought to the start of the level that they had chosen. The difficulty of implementing this approach is redirecting the game to the shop, and then to the start of the level without using the world map. A crucial requirement for this level would be to ensure quick loading to ensure that the gameplay experience is satisfying.

Fig 2: Concept image of Approach 2

3.0 SAAM Analysis

In order to determine which is the better approach to implement, the team conducted a SAAM (Software Architecture Analysis Method) analysis.

A SAAM analysis begins with identifying stakeholders of the software. Due to the nature of SuperTux being an open source game, typical stakeholders of games such as marketers, corporate officials and other investors are not present. The only significant stakeholders of the game are those who play the game, those who help in the development of the game (and the feature being proposed), and any program testers of SuperTux.

The two approaches to this feature will be evaluated by comparing them against some non-functional quality requirements the feature must meet. Whichever approach fares the best against them will be the approach used during the implementation phase. The actual non-functional requirements being looked at are labelled in Figure 3 according to the stakeholder associated with them. Figure 4 expands upon what the requirement actually is, and shows and advantages or disadvantages had by the two approaches against one another.

	Stakeholders		
	Player	Developer	Tester
Non-Functional Quality Attributes	User Satisfaction	Simplicity	Testability
	Performance	Extensibility	
		Maintainability	

Fig 3: Table of stakeholders and quality attributes

Non-Functional Requirement	Approach 1: mini level in world map	Approach 2: mini level before entering level
User Satisfaction (Player): The player should not feel hindered or inconvenienced by the store's implementation.	Player can choose to skip the store entirely, getting to gameplay faster Gameplay is delayed longer, player has to travel the world map to the store and back	If player wants to go to store, they can immediately go there at start of level The store is always loaded even if player doesn't want to buy something, delaying gameplay
Simplicity (Developer): Coding the implementation should not take longer than 2 days of work	Only one store in each world map, faster to code.	Every level has to be edited, longer to code.
Testability (Tester): Test cases written must check and resolve 95% of possible errors.	Store must be tested on every single world map to ensure it is working correctly. Limited number of tests.	Store must be tested on every single level to ensure it is working correctly. Lots more tests.
Extensibility (Developer): Implementing the feature should not impact the game's existing functionality	World map loads store, and store returns to world map like a normal level, minimal impact	When loading a level, the level will first redirect to the store, and then the store will return to loading the level, significant edit to level loading process
Maintainability (Developer): Coupling	Each world map has a unique call to the store, increasing coupling	Each level has a unique call to the store, vastly increasing coupling
Performance (Player): Loading the store should not take longer than 1/5th the time is takes to load a normal level	The store itself is essentially a miniature level, and will take a fraction of the time needed to load a normal level. Data for the store itself is programmed in the same way for both approaches.	

Figure 4: SAAM analysis of the two different proposed approaches

3.1 User Satisfaction

For user satisfaction, it ultimately came down to a matter of user preference to decide which approach was better. Does a player prefer the choice of travelling to and from the store or not, or do they prefer the convenience of the store loading before every single level? A

player who rarely uses the store will prefer approach 1. A player who often uses the store will prefer approach 2.

3.2 Simplicity, Testability, Maintainability

Simplicity was evaluated by how long it would take the developers of the feature to complete it. It is a relatively simple feature to implement, and so the team approximated it would only take around 2 full workdays. While both methods can feasibly achieve this, approach would take longer than approach one because every single level in the game would have to be edited to include a call to the store. Whereas in approach 1, only the world maps need to be edited to include access to the store.

The comparison results for testability and maintainability were similar. Approach 2's increased amount of calls to the store in the code (because of the way the approach is designed) means there will be an increase in coupling, and test cases will have to cover even more calls than they would in approach 1, making testing more difficult.

3.3 Extensibility

Approach 2 had a comparable disadvantage with extensibility as well. It is intended for the player to be redirected to the store when they select a level to play, instead of actually going straight to the level as normal. Upon finishing their business in the store, the player is then redirected to the level they selected to play. This does not preserve the original game design.

Approach 1 includes the store as essentially an extra customized level in the world map that the player can access through special means, and thus keeps most of the game's original function intact.

3.4 Performance

The store itself is essentially a miniature, customized, unique level stored in the game data. Thus it can be assumed it would take far less time to load up the store to interact with than it would an entire standard level.

The actual design of the store and how it is developed remains the same for both approaches; only the method in which it is accessed changes. It is highly unlikely there would be a performance difference between the two approaches, and if there were it would be so miniscule of a difference to be negligible.

3.5 Chosen Implementation

For the two approaches we chose, the only two requirements that really influenced the decision making process were User Satisfaction and Simplicity. While the approaches had some advantages and disadvantages against each other for other requirements, the had very little effect on the game and its development process.

The biggest thing the team took away from the analysis was that having the store as a customized level in the world map meant the player had more freedom to choose whether they want to access the store. As opposed to the alternative approach of having the store at the beginning of every level, in which the players don't have any choice.

The first approach is also simpler to develop. The main reason being it simply has less places where the store is called on from.

For these reasons, our group decided the best decision was to implement the store as a customized level in the world map.

4.0 Impact on Architecture

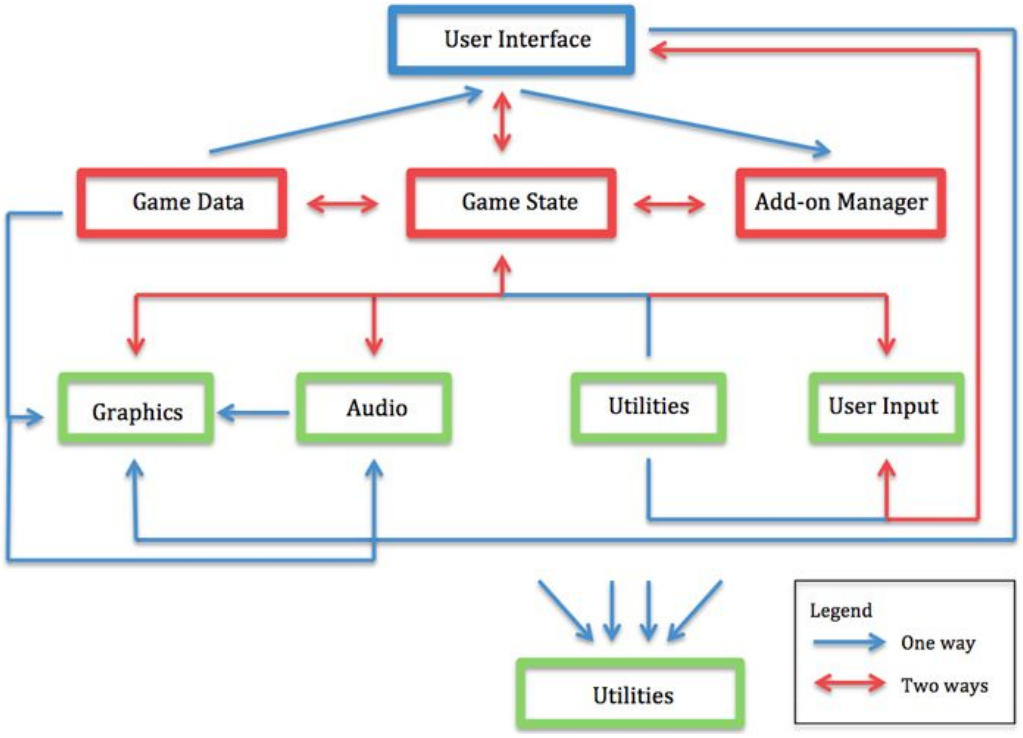


Fig 5: Concrete Architecture

The shop we implemented was a subcomponent of game state. Not all the functions we used were new, some of them were belong to the original C++ file.

Impact of Dependency

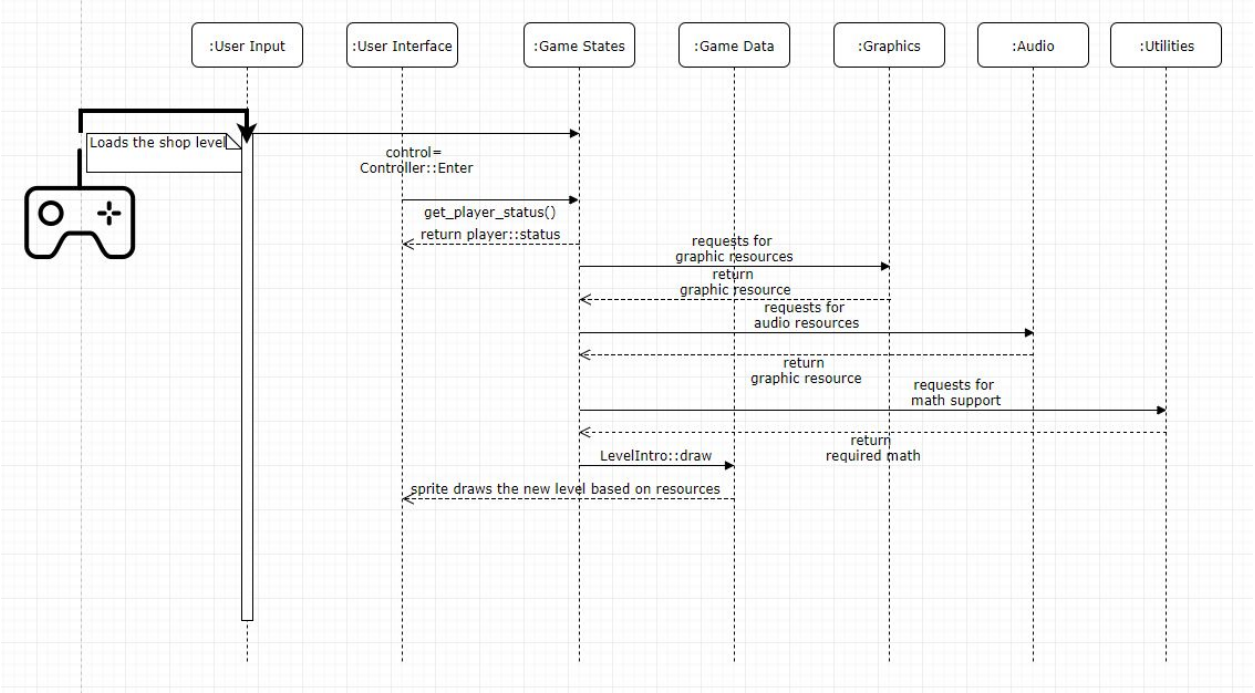
We added dependencies between our shop and the resource subsystem. We did not really change the components in resource subsystem, but called some of the functions in the files in order to implement the functions we need to implement. For the utility, the extra physic and mathematic functions were not added, just use the original features. The shop also had a dependency on the sprite which was a subcomponent of game data.

There was only one-way dependency between our implementation and these three components (audio, utility and game data), which means that if these free files were changed, our shop would be affected. However, if our shop was changed, no effects would be appeared on these components.

Changing

Some of the images of power ups were changed, this lead some changes on graphics. The user interface was also changed for a little bit, because a new room was implemented for the shop. The player would also use a new input to enter to the shop in world map.

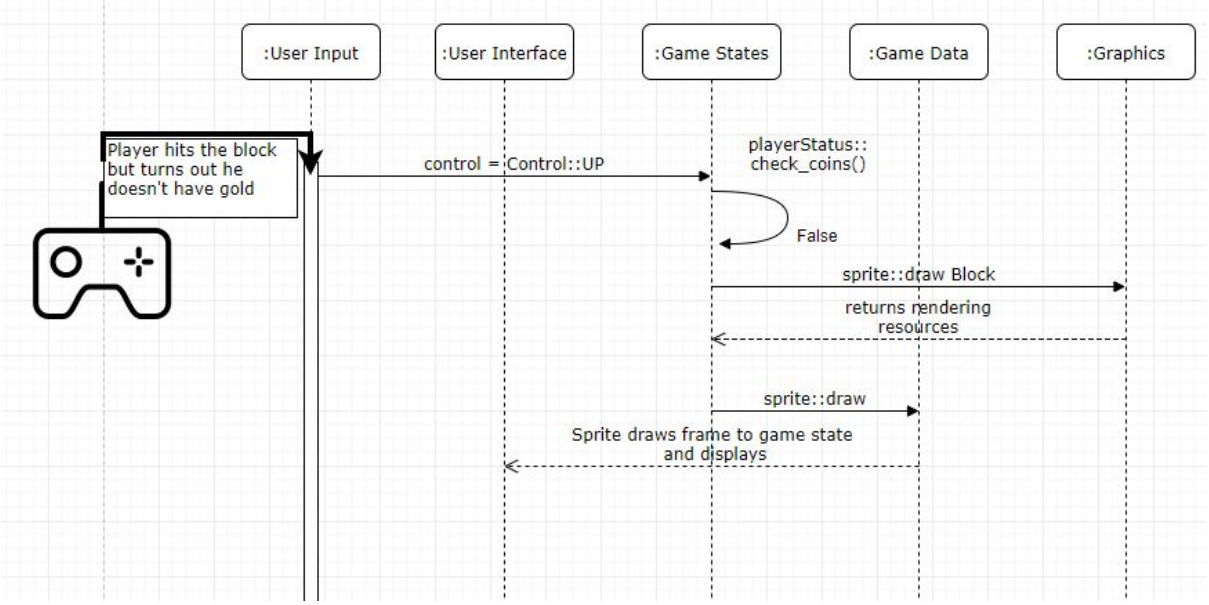
5.0 Sequence Diagrams



The above diagram describes a sequence of events when the player enters the shop level. User input takes the key press action from the player and sends a signal (control = Controller::Enter) to the current game state. This will result in a chain of calls to the game Graphics, Audio and Utilities system for level generation required materials. After the required parts been returned to the game states, game states will call the function Level Intro:: Draw method in order to start drawing the entire level with the graphics, maths and audios resources. The sprites inside of game data subsystem will be in charge of drawing the level frames and send to the user interface via game states. The user now can see the new level in his display.

The following diagram describes how a player tries to buy a snowball power-up from the shop but turns out to have insufficient funds. When the player doesn't have enough coins to

spend, the method check_coins() will return false immediately and causing the bonus block to act like a normal block, which contains nothing inside.



6.0 Risks & Limitations

For risks of this implementation, the team considered about user enjoyment so the player might think that a shop makes the game too easy and feels boring during the gameplay. Also, there might exist some potentially unexpected side effects following the implementation. For example, the implementation might lower the maintainability because of each world map has a unique call to the shop that increasing coupling.

For limitations of this implementation, the team did not have enough time to clear the game so that it is possible the implementation might not work in some levels. Also the team put some shops in the very behind levels and they were not tested so they might have some errors.

7.0 Testing

The team planned to test the shop as a standalone feature that is as a level in the game. The team must test the shop from entering the shop on the worldmap to exiting the shop and choose the desired level with power up works in that level. Also the team must ensure coins are deducted properly and correct power ups are received if there is no enough money, ensure nothing showed up. The new audio effect must work properly, the team added the sound for deducting coins.

8.0 Concurrency and Team Issues

Concurrency

The implementation of the shop turned out to have no effect on the concurrency of the game as the enhancement can be considered as a mini level. Just like the concurrency analysis for the concrete architecture, the concurrencies seems to be everywhere. Almost everything runs concurrency and this makes sense because the game needs to calculate each components and run quite fast for customer satisfaction, and if they were not delivered to the screen at the same time, components would lose their meaning. Some examples include, Audio and Graphics - the audio complements the graphics component, and vice versa. The source code includes a thread queue file that is used to handle the concurrency within the game.

Team Issues

Moving onto the team issues of the development teams, it's been found that finding the source code of the feature edited in the level editor is quite difficult. This is mainly because of absence of comments in the source code, which makes it hard to read and understand the code, and thus hard to work with. One of the the biggest problems is the permission required for every change made in the level editor. Also, communication between every member working on the code of this implementation as due to concurrency everything works together and slight changes may disturb some functionality, without communication.

9.0 Lessons Learned

Through our research, we have came to the understanding that every feature can be implemented in different ways. However, finding the optimal way to do so is highly important and depends on multiple factors. These include, but not limited to, reusability, ability to use pre-existing code and/or subsystems to implement the new feature, and time, the length of time required to implement the feature. It's important to note that, while some are very important, not all non-functional requirements provide insight when comparing different approaches, as we have seen during the SAAM analysis, earlier this report. Also, as mentioned in Team Issues, the use of comments in source codes are highly important as it helps understand the code, especially for people who haven't worked on the code at all, and it is useful for possible improvements, as nothing much can be done without a fair understanding of the code.

10.0 Conclusion

The objective of this report is to introduce and describe the new feature for SuperTux. The enhancement is a shop where the player can purchase various power-ups with their in-game coins. This feature will give the user the choice to the ease the difficulty of the level by using the shop. There were two approaches discussed, and SAAM analysis was done to compare the two approaches. The SAAM analysis showed that the most influencing non-functional requirements were User Satisfaction and Simplicity, and the chosen approach is Approach 1: The shop is a separate level in the world map.

For the impact on the architecture for this feature, certain dependencies were added between the shop and the resource subsystem and made sure the shop didn't affect any audio, utility and game data components. Also, the implementation of the shop led to some changes in graphics and the user interface. Sequence diagrams were added to provide an understanding of the enhancement.

There are also some risks accompanying with the implementation of this feature. For example, the shop having a negative effect on the user satisfaction, as this may make the game easier and/or boring to some players, and that there may be side effects due to the implementation. As a limitation, due to the time constraint the team had to work on this feature, the game might not work on some levels at the moment as the game wasn't cleared, and there may be errors due to non-tested sections of the shops in behind levels.

The shop is decided to be tested as a standalone feature that is just like another level in the game. What must be tested is entering and exiting the shop, and the use of powerups, and finally the deduction of coins and correct power up selection.

The implementation did not have an effect on the concurrency and still seems to be everywhere. Also, SuperTux handles concurrency within a game through a queue file that is found in its source code. The development teams need to be in great contact, as every change could affect another section due to concurrency. Also the lack of comments in the source code making it difficult to understand and write. Another team issue found to be is the permission request for every change made in the level editor, which our team has not been able to solve given the time limit.

Finally, few lessons have been learned during the implementation of this feature. Such as every feature can be implemented in various ways and because of this the optimal way should be found, and this depends on multiple factors such as reusability and time. Some of the non-functional requirements may not provide insight when compared between different methods. Also, comments in source codes are required for understanding of the code.

11.0 References

Implementation Description

We have successfully implemented the new shop feature into supertux by tweaking both the source code and the graphics resources by the smallest amount possible in order to keep the general structure of the game unaffected.

We have added a new method inside of bonus_block.cpp in objects. We named it to be lose_coins, it inherits the properties lose_coins from player_status.cpp files and defines a new modification to the current status of the player.

```
void
PlayerStatus::lose_coins(int count)
{
  coins = std::max(coins - count, 0);

  if(!play_sound)
    return;

  static float sound_played_time = 0;
  if(count <= 100)
    SoundManager::current()->play("sounds/lifeup.wav");
  else if (real_time > sound_played_time + 0.010) {
    SoundManager::current()->play("sounds/coin.wav");
    sound_played_time = real_time;
  }
}
```

Modifications to the power-up's behaviors have been applied in order to make the coins being deducted correctly only in the shop level but not in normal levels.
We made some use of a few unused/ defective/ unused power-ups, such as the unknown use potions. This is for the simplicity of putting the new power-up blocks into our shop level by level editor. In this way we do not have to edit the editor files to add in new textures.
We have added specific behaviors to those blocks in their functional definitions in order to make them act exactly like the target power-up that they are simulating (except they costs coins). For instance we added the line of calling lose_coins method(100) to the simulated block to deduct 100 gold coins when this power-up is obtained.

```
bool
PlayerStatus::check_coins()
{
  if(coins >= 100){
    return true;
  }
  else{
    return false;
  }
}
```

We added a checking mechanism (turn a power-up block empty) to ensure that player can't buy a power-up block with insufficient money. Finally we insert the changed power-up behavior code to the block. For example,

```
case CONTENT_AIRGROW:
    {
      if(!player->get_status()->check_coins()){
        break;
       }
      else{
        player->get_status()->lose_coins(100);// my adding
        raise_growup_bonus(player, ICE_BONUS, direction);
        break;
      }
    }
```

This used to be a EARTH_BOUNS.(Earth bonus never appear in the game except for the level editor)

And the final step we've done is to change some of the reference to images and audio files. Although we have saved some work from avoiding changing the editor file, we have to override the old unused texture by the new textures we want to use and the sprites used to draw the replaced power-up effects.

After implementing the code and texture, we added the shop level into world map via two ways. First, I thought the level editor was not able to change it, so I manually added the code representation of each texture of paths and teleportation ports to the tiles machine language file, soon we found out that it could be done in the level editor, so we modified them a bit and made a shop level with the editor.

With the already implemented save feature of supertux(savegame.cpp), the power-ups and coins obtained from last level will be saved to player status automatically. So to this point we have successfully implemented the feature without problem.

We then compiled the whole game with modified code, and tested with use cases, it seems to be working properly without errors.

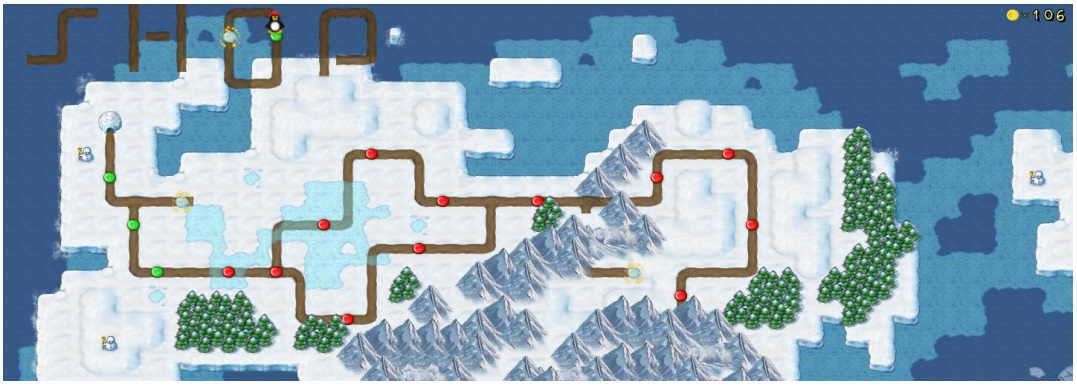


Fig 3: world map with implementation

Fig 4: Implementation shop: the blocks from left to right is a free egg, a fire flower and a ice flower but there is no enough money so the ice flower did not show up. The other blocks are only for testing and decoration purpose.

Git: https://github.com/nbcstevenchen/supertux

little problem about uploading source code files: it seems like the supertux game has a weird permission error about finding the saved changes to the file if we edit or create a level in level editor. So that we are unable to find the shop.stl (the shop level created by editor that showed in Fig 4) file anywhere with the changes we made by the level editor. Unless we play the game in contrib levels instead of story mode. (Contrib levels can connect a level red point to a level from level editor directly without knowing the location of that file) In that case, we are unable to upload the shop.stl along with the source code. But all other changes has been packaged without any problem.
We can arrange a time for another demonstration to TA or make a short video about our implementation if needed, if the last times demo is not sufficient.