netCademy

Collaboration Games

TDT4245 Cooperation Technology and Social Media

Final Project Report

Group 09

Lars Føleide, William Henriksen, Sindre Skarås,
Magne Skjæran and Vegard Theriault

Introduction

This is the project report for the course TDT425 Cooperation Technology and Social Media.

The purpose of this project is to formulate a game concept and produce a prototype or an early implementation of a game meant to teach teenagers an aspect of IT development through collaborative processes.

Learning objectives

Learning objectives	● LO1: Learn how collaboration can achieve goals that could not be completed individually ● LO2: Learn how to build shared awareness of the workspace when there is information asymmetry ● LO3: Learn to communicate in an effective way to overcome the constraints of the in-game tasks ● LO4: Learn the basic elements of programming and programming logic ● LO5: Learn how players can coordinate work based on their different work capabilities
Genre	Two-player puzzle game, casual in a 2D CVE (Collaborative Virtual Environment)
Brief description	Two users have to cooperate to achieve a variety of tasks on a 2D map. This is done using one in-game character each that are programmed by the users. On different levels different constraints are applied such as information asymmetry, different abilities, and other things that force the players to collaborate and communicate in different ways.
Gaming activities	● Communication with the other player to establish a common understanding of the task at hand ● Programming of characters ● Achieving coordination of on-map actors, including timing considerations
Constraints	- The game can be played in short (30-45 minutes) time spans - The players are not allowed to see one another's screen
Collaboration	The two players are located in the same room, allowing them to talk to one another

Target group	The game will be mainly targeted at middle school students with no previous experience with IT development

Achieving learning objectives

We came up with a set of five learning objectives that we want our game to help the user achieve. We've numbered these LO1 through LO5.
This section details how the game will support these learning objectives.

LO1 - Learn how collaboration can achieve goals that could not be completed individually

Most tasks in our games will be either extremely difficult or outright impossible to complete without cooperating with the other player. As an example, in one task only the first player might be able to view the map, while only the second can actually interact with it. Without communicating with the first player, the second player would be working blind, while the first player simply cannot complete the task except by helping the second player complete it.
By effectively forcing the players to cooperate, we hope to ensure the players acquire a better understanding of collaboration, and how it can often be more effective than working separately.

LO2 - Learn how to build shared awareness of the workspace when there is information asymmetry

The tasks in our game will often involve some degree of information asymmetry. The previous example of one player being blind is a very clear case, while for most tasks it will not be quite this extreme. The players might for example have different starting positions, and thus see different things. Some tasks will use a line of sight system, meaning that if players are in different positions they'll see different parts of the map. Other tasks will have the players seeing different "layers" of the same map; one might be able to see electrical wiring under the floor for example, while the other would see it in the roof.
By introducing these forms of information asymmetry we ensure that the players must communicate in some way to one another in order to complete their tasks, making them establish a shared awareness of the workspace.

LO3 - Learn to communicate in an effective way to overcome the constraints of the in-game tasks

In order to complete the game's tasks, the players will need to communicate. Effective communication is further rewarded, as that will help them complete the tasks more quickly. The two learning objectives above further reinforce the learning of effective communication.

The game will also have a few features assisting communication. Most importantly, there will be a “ping” feature, that allows one to indicate a specific position on the map. As the players cannot see one another’s screen, this is essential to ensuring the other player actually understands what you want them to do.
Second, there will be an indicator showing what direction the other player is in if they’re not currently on-screen. This will assist the players in being aware of one another’s actions.

LO4 - Learn the basic elements of programming and programming logic

The agents that the players of our game will play as will be programmed rather than controlled directly. This will be done through a drag and drop interface, similar for example to the programming language Scratch. The players can insert blocks like “move”, “turn left”, and “wait for event X to happen”. The players then click “run”, to see if their routine solved the current objective. If the routine described by the players doesn’t succeed, they will have to start over. They can then set breakpoints (a point in execution where the execution will pause), and step through block by block in order to figure out where they went wrong.

LO5 - Learn how players can coordinate work based on their different work capabilities

The players will have different special capabilities such as running faster, jumping higher, or unlocking specific locks. This means the players will have to coordinate who does which part of the current objective, based on those capabilities.
The players are also likely to differ in how quickly they’re able to understand and master the game. Our hope is that by communicating they’ll be able to boost one another’s learning by helping each other in the areas where one does better than the other.

Other supporting elements

We’ve taken a number of other steps in our design to help ensure a good learning experience. For one, we’ve considered the six guidelines presented in[1] “effective game design for learning”.

First of all, Whitton suggests that the game environment should support active learning. We’ve tried to achieve this by making the players’ effect on the environment clear, and presenting them with problems that can be solved as long as one thinks them through.

Second, the environment should engender engagement. We try to achieve this by giving the players explicit goals on each level, but leaving how to solve it up to them.
Our programming-based approach means that each level will have many possible solutions.

Third, the gaming world should be appropriate for the learning context. The main form of interaction in our game is programming, thus teaching the players the basics of programming. By having the player characters be robots, we have a clear narrative connection to the learning objectives.

Fourth, the environment or associated activities should provide opportunities for reflection. This aspect we've left out, as we consider "associated activities" to be outside the scope of our work so far.

Fifth and sixth, the environment must provide an equitable experience to all users, and provide ongoing support. We try to achieve this largely by having the game start off easy and then gradually get more challenging, with the players allowed to bypass levels if they're unable to complete them. This we hope will ensure that players of all skill levels can enjoy our game. This should ensure that players are able to feel like they're good at the game, as also mentioned on page 143 of Whitton (2002).

Functional specifications

Our functional specifications are based around the table presented in Whitton 2009[2], page 142.

	Functionality required
Environment	Players need to be able to see a map of the environment, with the artifacts present Players must see the result of their program by the character going through a set of animations and movements The game is split into levels, each level has its own environment The level environment will be split into a grid, to simplify character movements and object placement When executing the instructions, the time of the environment will be sequential counting turns, each instruction taking a number of "turns" to complete Players need an interface view for the programming, which houses instructions available (toolkit) Players need a view where they can drag and see the instructions that are about to be executed
Navigation	Players must be able to navigate between the programming view and the execution view by starting and stopping the instruction execution

	During execution, the characters must navigate and interact with the environment according to their instructions
Tasks	The players will be informed about a common task at the start of a level The task may also be visualized by the objects showing their incomplete status In most levels, each player will only see what their character sees, and must share their awareness with each other The players must agree and collaborate on how to solve the level by communicating out- or in-game and by trying their instructions
Characters	Each player will receive a character with some recognized abilities and restrictions. One player's restrictions will usually be directly supported by the other's abilities The character will act as a the vision for the player, allowing them only to see the objects that are within the character's line of sight (see view-blocking objects) - Stubby: Small, can only see low objects (even if concealed from above), but can perform demanding tasks (breaking boxes or carry heavy things) - Slinky: Flying, can see objects and the environment from up high, but can't get down on the ground.
Objects	The game will have a number of objects in the level environment that will either work as an aid, a challenge or a completion requirement to the players' task - Lock: Makes an object unavailable to carry or be interacted with until unlocked - With chain: The chain will cover the object that the lock makes unavailable for the players - Key: Carriable, can be carried to a lock to unlock it. - Door: Can be interacted with to be opened, allowing the character to go through it - Ladder: By interacting with this, a character will climb to the top - Cat: Can be carried, and may be just the thing a certain cat owner is looking for - Quest giver: A person that offers a quest (aka. mission) that must be completed as part of the task of the level. Will have an talk bubble over their head that shows the quest they offer the players. - Cat owner: Shows a cat stuck in a tree, hinting that the players must retrieve it to them

	- Blocking objects (walls): These will stop the character from walking to a spot, by acting as something with collision - View-blocking objects (trees, clouds): These obstruct the vision of the character, breaking their line of sight to certain objects and details, which carries over to what the player sees Instruction objects (listed below): These are the instruction boxes available for the player to drag and drop inside their programming views
Object interaction	Each object will be of different types, where some are interactable in the execution phase by characters standing next to them and using a corresponding instruction. - Carriable: Certain characters will be able to pick up these objects by using the Pick Up Object instruction - Useable: Such objects will have a function that is performed when a character uses the Interact with Object instruction - Useable and Carriable: These objects may be used while carried, which is done by the character executing the Use Item Instruction
Player interaction	General: The characters interact with each other indirectly by affecting objects in the environment, which impacts what the other player can do. Example: Dropping a key for the other to pick up Ping: The player can communicate to each other by clicking with their cursor (or finger) in the environment, which will show a “pinging” marker at that spot, which is visible to the other player and can draw their attention to a spot Send code: By dragging one of their instructions from the programming/toolkit window to the environment, the players will be able to simultaneously see that instruction floating there for a small while. “Wait for other player” instruction: This will make them wait for the other player to use the same instruction before continuing
Status information	Interface shows status of the session (name of player they’re collaborating with) The program has several of the program (instructions present, number of instructions, etc.) Several instructions has a value that can be set - this will be shown as status on the instruction

	Many objects will also have a visible status: - Quest giver: Show status in their speech-bubble - Lock: Shown as open or locked - Door: Will be open or closed

Programming Instructions

These are the instructions that the players can drag and drop from their toolkit to control their robots. Note that only a subset is available in each level, based on what is needed and what has been introduced in earlier levels.

Robot control

This is the meat of the game. Each of these instructions make the robot do something in the environment.

- Movement:
  - *Walk <left/right>*: Moves one square in the direction
  - *Jump <up/down>*: If there is a platform over/under the robot, this will make it jump up/down to that floor
  - *Climb <up/down>*: Climbs up or down one square, if on a ladder or similar
- Actions:
  - *Interact*: Makes the robot activate/use or otherwise interact with an adjacent object in environment
  - *Pick up object*: Picks up a single adjacent object, if carriable
  - *Drop object*: Drops the currently held object
  - *Use item:* If holding an object that can be used - such as a key - this will make the robot try to use it where it stands
- Misc:
  - *Wait for other player*: Makes the robot wait until both players are using this instruction at the same time before continuing on to executing the following instructions

Programming

These instructions are more internal, to control other instructions or to help the players solve the task or create complex programs more easily. Since the actual control of the robot characters is difficult enough, the game will mostly have the players use simple programming instructions.

Simple:

- *Repeat <times>:*
  - Repeats the given set of instructions a certain amount of times
- *Wait-for <turns>*

    - Waits for a while (given by “turns” in the game) before continuing on to the next instructions

The following are other instructions that later levels might make use of, to introduce more complex programming tasks to the players.

Advanced:

- Variable assignment
    - Allows storing of data for use later
- Operators: Arithmetic (+, -, *, /, %), Boolean (equal, not, <, >, etc.)
    - These are used to compare or change two variables. The result can then be used in an “if-then-else”-instruction
- If-then-else
    - Executes one set of instructions if the received value is true,
      example: ‘If “x > 2”, then Move Right’
- Loops (for, while)
    - Repeats a set of instructions several times until a value becomes false
- Functions
    - Allows storing several instructions for reuse in a new, composite instruction
- Goto
    - Jumps to another point in the instruction list

Game Description

Two users have to cooperate to achieve a variety of tasks on a 2D map. This is done using one or more on-map actors that are programmed by one or both of the users.

On different levels different constraints are applied such as information asymmetry, different abilities, and other things that force the players to collaborate and communicate in different ways.

Plot

Robot Rescue Patrol: Guide the Robot Rescuer Rangers on their job to help those in need and perform good deeds. They need their instructions to know how to act, and the two of you are the ones we need to instruct them! To achieve this goal it is imperative that you work together, as no single robot nor instructor can do the job on their own. Go through levels of different task and rescues, where the tasks become increasingly harder, and you will find new ways you must rely on you instructor partner. Save the robot cat from a tree, help the drowning robot swimmer, or simply guide the robot grandma across the robot highway.

At you disposal are these top-of-the-class Robot Rescue Rangers:

- **Stubby**: strong and stocky, but struggling with a height complexity: He excels at getting down and dirty with the heavy lifting and manhandling his way through the job, but don't rely on him to scout ahead or get to high places without the help of a certain partner.

- **Slinky**: swooping above on her own cloud, she is a shining scout and and can touch the heavens (metaphorically). Don't rely on her to ever come down, though: Her cloud has a will of its own when it comes to height, preferring to stay exactly where it pleases.

Platform

Our platform of choice for this game is based on touch technology. We've chosen to employ touch as it is conducive to collaborative learning[3], the technology can be implemented in several ways that can be useful for our game:

- Tablets
- Tablet computers such as Microsoft Surface
- Desktop computers with touch capabilities
- Laptop computers with touch capabilities
- Multi-touch tables[4]

The younger a person is, the more likely it is that they're accustomed to tablets and similar touch technology[5], and since we're aiming for an age group comprised of youths, touch technology is the logical choice for our platform.

The hardware involved is not especially important, as our game should function similarly across several type of touch-capable hardware.

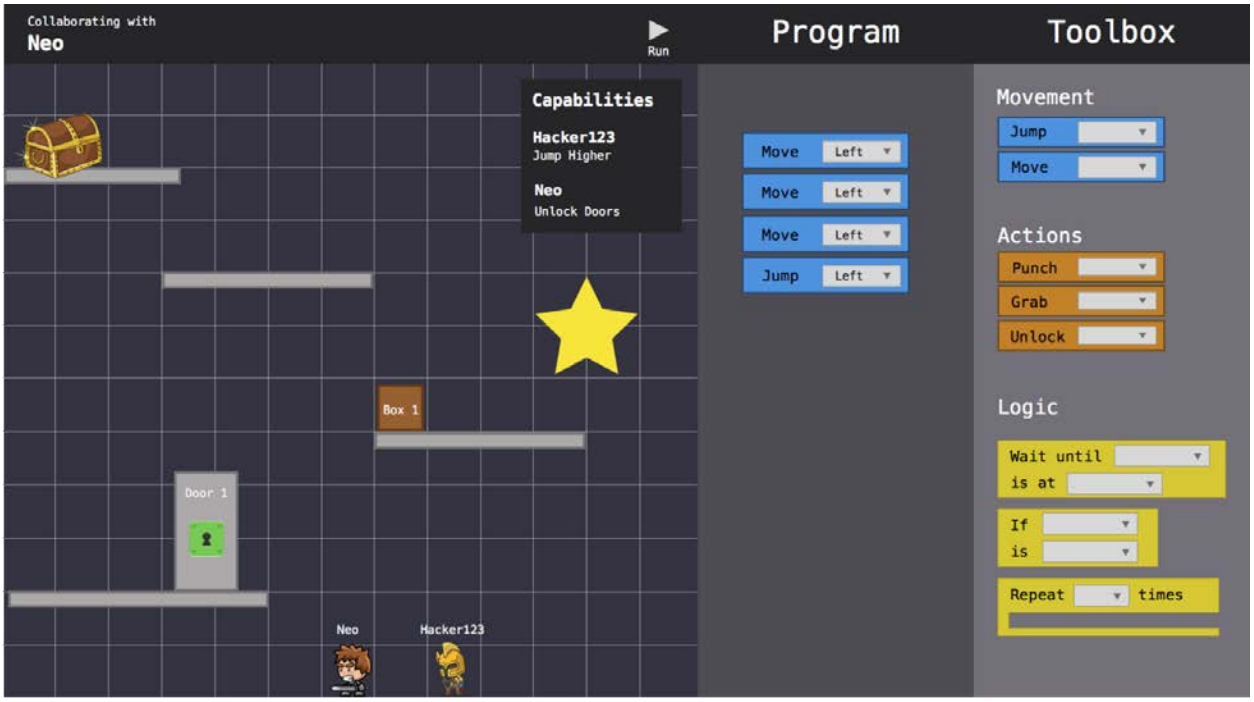
Underlying technologies

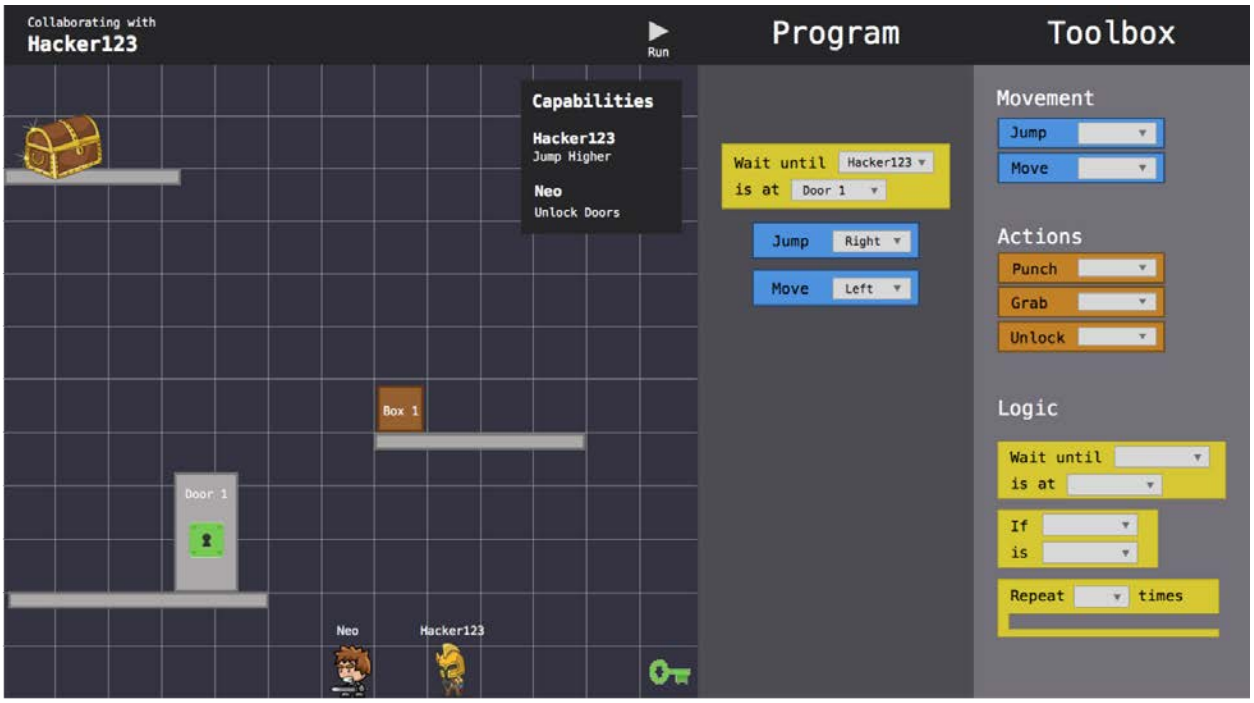
For this project, the most important aspect to consider in relation to technologies is that the two players need to see the changes done by the other player. For this purpose, we introduce a SQL database which both clients will communicate with through PHP.

The application itself will be made in the Unity game engine, which is a framework for developing cross-platform games. This means that our application would be able to run on several different hardware platforms such as Microsoft tablets, iPads, conventional computers and others.

Demo and Scenario of Use

Early Concept Illustration



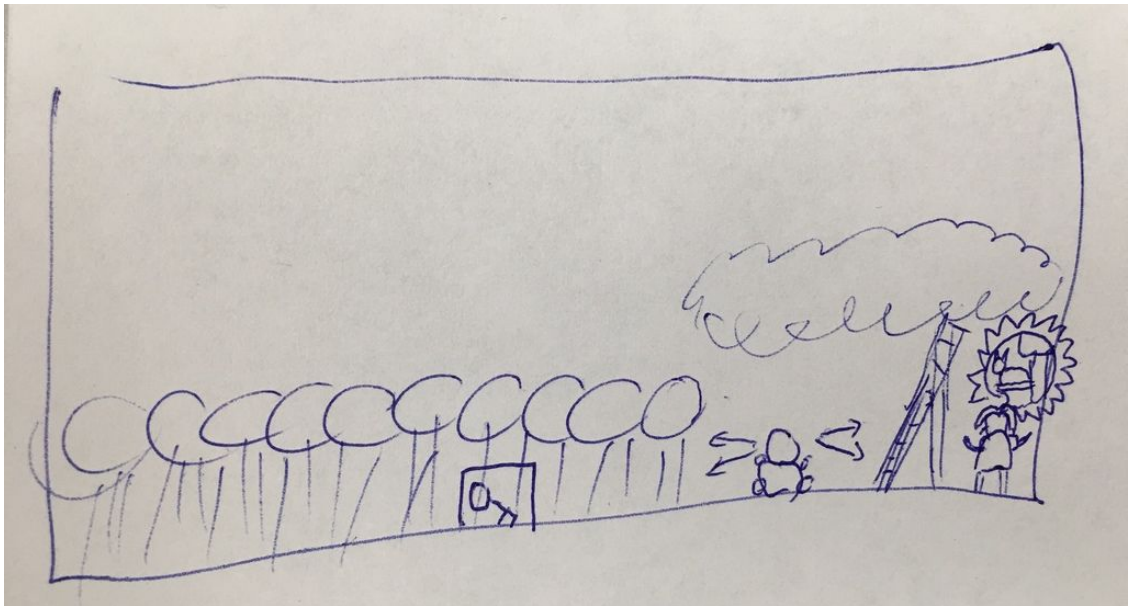


Our early concept illustration show Neo and Hacker123 needing to collaborate in order to solve the level, as they have different abilities. At this point, all that was clear is that we wanted them

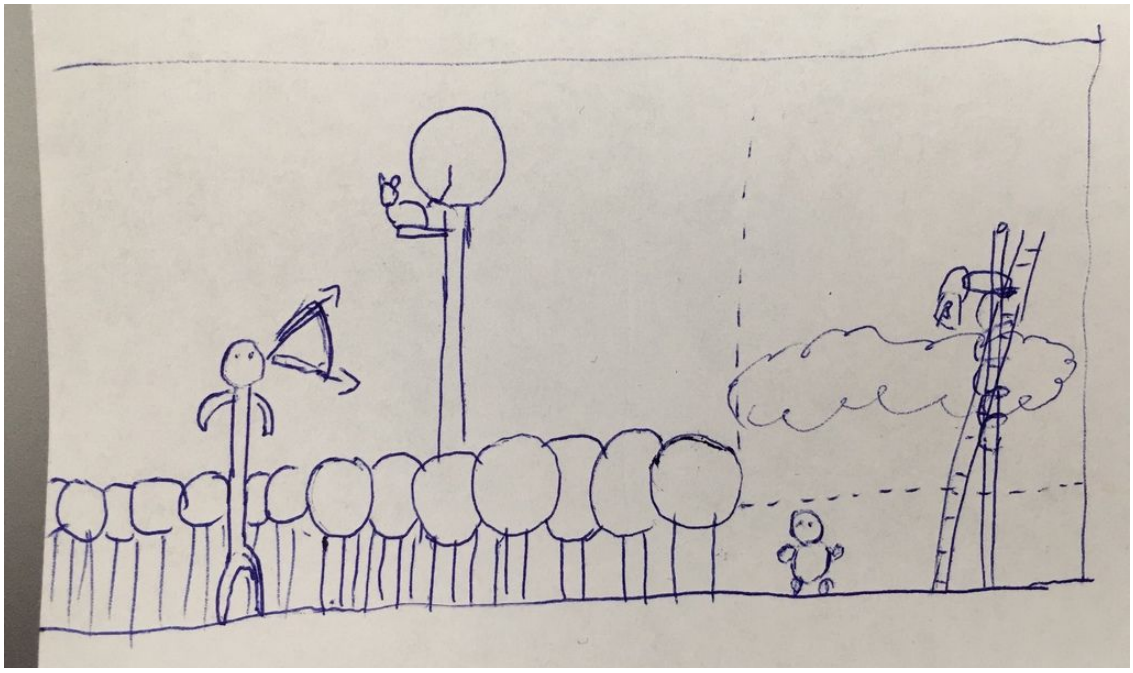
to see different things, be controlled by instructions on the right, and to move in a 2D sidescroller environment.

Low-Fidelity Demo Sketch

Player 1 (Small Robot: Stubby)



Player 2 (Tall Robot: Slinky)



These two Low-Fidelity screens illustrate the views of the two players before we digitized it into the high-fidelity demo. The different elements visible to the players are drawn using dashed

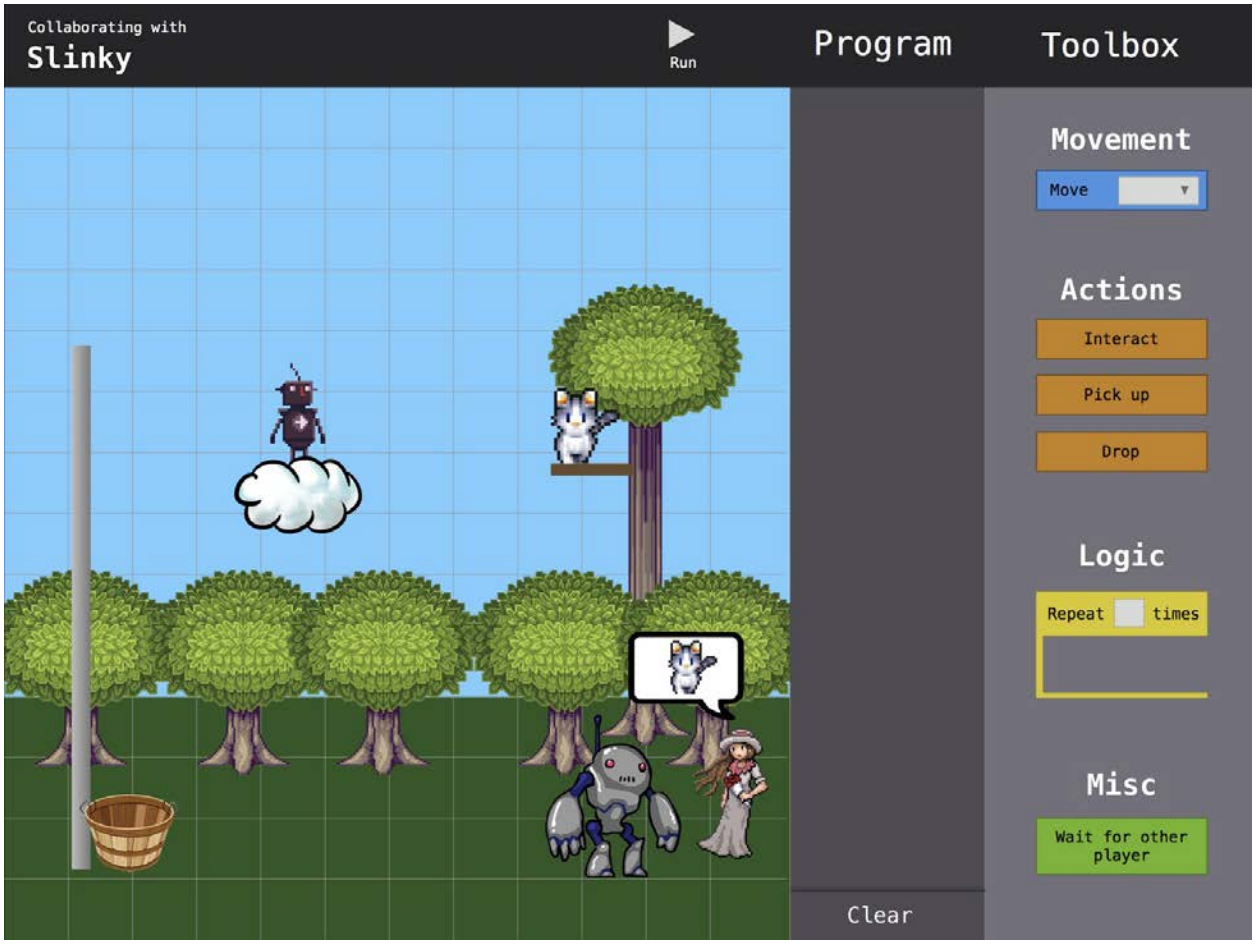
lines. At this point we had an idea of everything in the concept, but some changes was still made to the next version: The level was simplified, with a reduction in steps, yet not too much, as we still wanted both players to have something they could see that they had to communicate (quest giver for Stubby and the cat for Slinky), and also had each an active responsibility in getting the cat to the quest-giver / cat owner. Slinky was also changed from a tall robot to a cloud-riding one, to minimize its screen space and make its limitations in the game clearer.

High-Fidelity Demo

Level: Save a lost cat from the woods

A girl is shown, missing her cat. It is clear from her speech bubble that the cat is stuck in a tree in the woods.

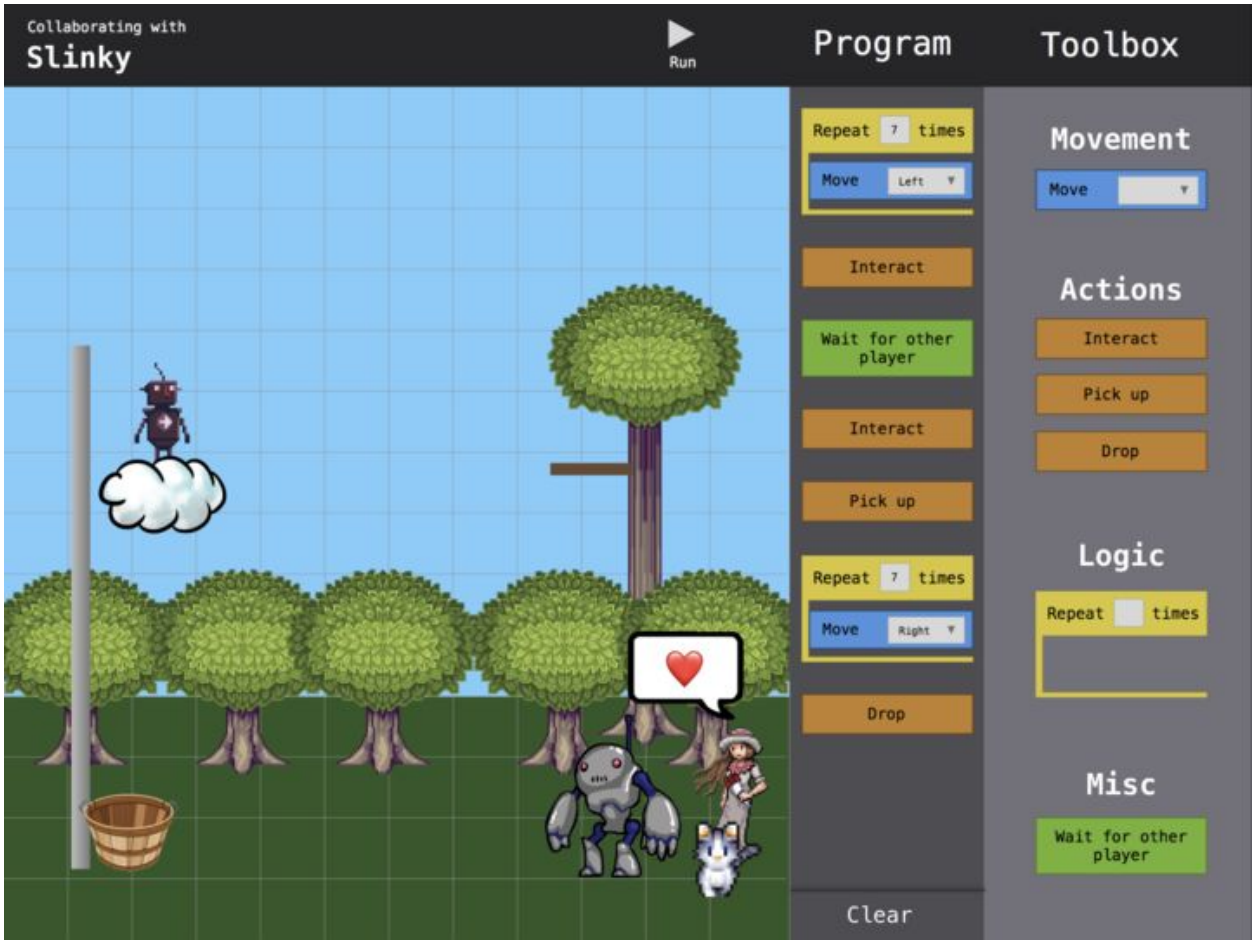
These pictured show the difference in views between the players controlling Slinky and Stubby. Stubby is the one on the left, and Slinky is to the right.



The robots Slinky and Stubby are here given the task by a girl (quest giver) to save her cat. It is stuck in a tall tree, so Stubby on the ground needs to collaborate with Slinky in a cloud so that the girl can recover her cat. The two players are presented different views, Stubby cannot see above the trees while Slinky cannot see below the trees. So they have to communicate and collaborate in order to solve the level.

On the far right, our robot player Stubby (who's collaborating with Slinky) is presented a toolbox with various instructions for the program. Through drag and drop the player controlling Stubby can build their program to retrieve the cat in collaboration with Slinky.

Stubby has to move 7 times to the left for reaching the basket that can be sent to Slinky. Instead of inserting 7 move boxes, Stubby can insert a logic function that repeat the same move 7 times.



Stubby is waiting for Slinky to put the cat in the basket so that he can move 7 times to the right and give the cat to the girl. Upon retrieving the cat, the girl expresses much joy of having recovered her cat.

Demo: Toolbox and Programs for the players in this level

This is the toolbox available to the players for this level:

- Toolbox:
  - Movement
    - Move left
    - Move right
  - Actions
    - Interact
    - Pick up
    - Drop
  - Logic
    - Repeat X times
  - Misc
    - Wait for other player

These are the actual programs that we expect the players controlling Stubby and Slinky to develop in order to complete the level:

Program Stubby:

- Repeat 7 times
    - Move left
- Interact
- Wait for other player
- Interact
- Pick up
- Repeat 7 times
    - Move right
- Drop

Program Slinky:

- Wait for other player
- Repeat 4 times
    - Move left
- Pick up
- Repeat 6 times
    - Move right
- Drop

Upon completing the level, our two character will be presented a new challenge. As the game progresses, our players can be presented new functions in the toolbox. That will ensure a natural flow in learning new programming concepts.

Learning IT development

As seen in Learning Objective 4, one of the main goals of this game is to teach a basic form of programming. This is the key element of how we aim to teach IT development. The other focus is in regards to cooperation, which is covered by the rest of the learning objectives. The players will work together to achieve a task similar to IT-development along with numerous ways to stimulate and support the cooperation, which are presented below.
The main way this is achieved is through the game mechanic where each player has to program a robot to complete the task at hand. This provides an introduction into rudimentary programming logic and concepts, as well as introducing the user to the idea of debugging and how useful debugging tools can be via the use of breakpoints and the ability to “step” through the execution of their code.

This uses a drag-and-drop interface reminiscent of the programming language Scratch, to make it easy for players who have never programmed before to learn the basics. This does place some limitations on what can be taught. For example, this approach cannot teach the player how to structure a large program, as the syntax is simply too unwieldy for anything beyond small programs. It can however still teach basic structure such as the use of variables and functions. The core logic itself can be learned, as the syntax is well suited to things like loops, functions, and arithmetic, which are all things the player will then be able to apply more easily in traditional programming languages.

Cooperation support

Each of these mechanics supporting cooperation use the respective slides on Communication, Awareness, etc. as presented in the lecture.

Communication

Communication is supported primarily through in-person verbal talking. This will be (intrinsically) informal and synchronous. The game will also allow the players to communicate using in-game actions and capabilities. The players can see each other, and the environment will be shared. They are also able to gesture to each other by clicking on the environment, producing a flashing “ping” indicator visible to both. By dragging a piece of code from the instruction set to the visual window, the players can communicate pieces of code, which will be inherently reviewable before sending. This instruction will then float there for a while, allowing the players to communicate using this instruction, both through the instruction as an artifact, and through its position in the environment. For example, one player can drag an Interact instruction on top of a button, while saying “Hey, use this instruction here” to the other player. These are all synchronous, informal types of communication, and we plan for the game to teach and encourage the effective use of these.

As for affordances, the premise of the game is to simulate difficulty by neglecting to give the players some essential affordances. These will take form of different world views (visibility - different elements, audibility - some different audio clues, and tangibility - can touch/interact different things). In the demo level, one robot is tall and can see/interact over the trees and clouds, while the other is small and sees/interacts below. The communication will also be supported by affordances in terms of some co-presence/visibility - knowing they share the same environment, as well as some similar visibility of the same elements and each other. Other factors that help enhance the communication will be the ability to move and show your movements in a sequential, asynchronous fashion, wherein a programming phase is followed and interchanged with a movement/execution phase.

Awareness

Now, teaching the players how to attain shared awareness is a another key component in this game. The game will direct the players to impart their awareness with all the available forms of communication - talking in-person, pinging on the map, and other context-sensitive interactions. The player will be taught to gather additional information from both the artifacts/environment, their mutual conversations, as well as the effects of each other's actions on the environment. Artifacts in particular will contribute with info such as task criteria/progress (quest person) or status about itself (locked/unlocked ladder, open/closed doors). Specifically, a quest person (see Functional Specifications) will give awareness in the form of their state: If their quest is unfulfilled, their speech bubble shows a picture of the objective (in the demo: retrieving a lost cat). Once it is completed, their demeanor will change to a happy state, and the bubble will disappear. Because the game is about robots following instructions, it will in practice offer a history of movement, actions, and resulting events, which grants another source of awareness.

By having different views and abilities, both of which are needed to attain the solution, some interdependency is created between the players. This forces them to engage in sharing awareness both about what is present, but also what they can do to help each other's obstacles and possibly the goal of the task.

Coordination

Some coordination will already be inherently supported by the format of a programming phase, where players will be naturally inclined to discuss and plan the task. In addition, this coordination will build upon what has already been mentioned in the section about awareness and communication. The most important ones there are those artifacts which provide an indication of what needs to be done and how (quest person, lock). By keeping several such artifacts that the task depends on, they help split the task into sub-objectives, which handles complexity, and shows progress to the main goal. For example, the task may require the players to go through a lock, and then a quest person. The players' coordination is then supported by letting them discuss the task as a series of steps.

To help handle some complexity with time, the player get a “Wait for other player” instruction, which makes it so they don’t have to keep track of how many turns they they must wait. allows the players to

Finally, the different capabilities of each player’s character gives them different , which is another form of handling task complexity and consequently supports coordination.

Information sharing

We don’t see any big need to support information sharing: Some support of information sharing will come from the ability to display an instruction block to the other player. This will mainly be for communication, but should the information within the instruction be interesting (such as being an instruction only one player can own and see), this info will also be transferred to the the player. Despite not needing explicit support for this, the game will still rely on information sharing for collaborating and completing the task. The game will encourage the players to share everything they see and think, quite possibly with a tutorial.

Social interaction and Player community

By being a multi-user game enabling collaborative gameplay in the same physical space, our game fosters a fair amount of social interaction. This is especially important by letting us use the constructivist learning approach, which encourages learning by discussion, sharing perspectives and finding a solution in unison, as described on page 47 in Whitton 2009 [1]. Furthermore, in pages 38-39 Whitton identifies social interaction as one of the key motivators that makes (serious) games more enjoyable for players.

When it comes to player community, the game has no mechanics providing support. It will however, incorporate some support as a natural part of its instructions and tutorials: To further support the constructivist approach, reflection and discussion among the players to over what they learned in the game will be promoted in the game. Doing this in a larger group is further encouraged. This can be a valuable time for a small community such as a school class to have a group discussion, and to make sure the learning objectives really hit home.

Reflection notes

Project coordination

In order to coordinate our work, we used several technologies for communication, awareness, etc as specified in the section below. The actual process of coordination was done mainly

verbally in meetings, or on Slack. We performed a number of activities here, as detailed in the lecture slides:

For planning, we first discussed in meetings how to proceed with the work. We chose Google Docs for synchronizing our work, as we were all familiar with it and believed in the benefits it could provide us with quick awareness and easy commenting. In conjunction with this, we used Google Drive to share the actual documents. Here we placed the report template from It's Learning, the actual report document, as well as extra internal documents as needed.

Next, we developed a writing strategy. This ended up as writing the report mainly in unison, with each member picking a section at a time to work on. We made sure to share a vision and awareness of what these sections should contain by discussing their contents beforehand. The report template was in this phase integrated from a separate document to extracted parts placed in comments along each relevant section.

The actual allocation of resources and roles were done informally and on an ad-hoc basis, with each member picking the section they felt comfortable with. At the same time, the group would aid each member, to make sure everyone had a task they were confident with. Towards the end of the project this became somewhat more formalized, with a list created of all remaining tasks and each team member picking one, then a new one once finished, until all the remaining tasks were done.

Overall progress was initially not tracked in any formal manner; comments in the delivery document were instead used to point out sections that were not yet completed. With the formalization of the allocation process, the allocation document also became a way to track progress in a more consistent manner.

Early on we defined a simple, but not entirely concrete, set of success criteria for when a report was ready for delivery. Simply put, all members of the team had to agree that all aspects of the task had been addressed, and that there were no remaining complaints about any of the report's contents. This ensured that each report would only be delivered once everyone on the team agreed it was ready.

Scheduling meetings was done primarily on Slack, but also verbally at the end of each meeting. At the start of the project, we used Doodle to find ideal meeting times.

The coordination mechanisms used were in other words mainly Google Docs, which served as an artifact displaying real-time status, progress, and criteria. It also helped us with handling complexity by the mentioned division into sections, each of which was a sub-task. Slack and verbal discussions formed the other pillar of coordination, in the form of both formal and informal ad-hoc communication.

Technologies used in collaboration

At the very start in the semester, we relied on It's Learning as a way to gather the group and coordinate a first meeting. This was the only way It's Learning was used, as we much preferred other communication tools that were quicker (Slack) and richer (Google docs).

From here on out, we started a channel on Slack, where most out-of-meeting communication was done. This allows both synchronous and asynchronous communication, ensuring everyone knows what to do, and when to next meet. It also allows the communication to be divided into channels, so that different topics and conversations are separated. Further, it allows for uploading files, which we used to share Google Documents and images of the game. This covered our needs for collaboration through communication in terms of information sharing (helping each other with problems, sharing textual info), coordinating (planning meeting, simple tasks), awareness (making sure everyone knows what happens in a meeting. Finally, it also allowed for some general collaboration and discussion about the project outside meetings. Although as useful as it might have been, the textual communication and partly asynchronous nature of Slack made it way less efficient than verbal talking: It was a less natural way to express rich ideas, and answers could be sent hours if not days after a message, if the user didn't have the text message function activated..

The next technology that were only used at the start is Doodle. This is a easy way to find overlaps of free time in everyone's schedules. At the start, these overlaps were hard to find, but as the semester progressed, we could more easily stick to fixed a meeting time, making Doodle redundant. While valuable and easy to use, Doodle is very simple, and the information everyone had to put in could sometimes be not rich enough to represent their varying availability. The protocol of use was also not immediately obvious, resulting sometimes in different interpretations and opinions of usage.

For document sharing, the group uses Google Drive for distribution, and Google Docs for the actual collaborative writing. This allows simultaneous editing of documents, letting multiple members see and work on the document at the same time. It's also an excellent way to grant a shared awareness of work and progress. The comment functionality let us also it for communication and coordination, In other words, it supports all communication mechanisms, including information sharing. Some drawbacks still exist with the actual quality of the writing and produced document: It gains some ease of use by sacrificing more complex functionality found in other editors, such as ShareLatex. Among these functions we could have used are an automatic bibliography, image/artifact references, and more stylization tools which would have created a more professional look. We still stuck with Google Docs for the work, and decided against copying over into Latex, being mostly satisfied with the document quality that Google Docs delivers.

As we closed into the middle and later parts of the semester, we found that we needed some way to construct a mock-up of our game, and a way to create a demo of a level. First we tried some digital methods of drawing, such as simply using MS Paint. This worked poorly, as it was limited to one screen, and was too difficult to quickly produce an illustration that was recognizable to the rest of the group. Instead we opted for simple pen and paper. This had an immediate improvement in that everyone could more easily see and be aware of what was being drawn. It was also quicker to make, and when presented, offered more affordances in terms of tangibility, with members pointing and moving to communicate different concepts. Finally, was easier to share by physically handing it to someone or even taking a picture to share on Slack.

We chose to create the demo using Sketch on Mac. Using this, we could make vector pictures using imported images and animated these as if they were moving in-game. It also supported interaction, which led us to make a pseudo-functional prototype that played the correct action in a sequence as the demonstrator clicked on the correct buttons. We also used this program to create a demonstrator of ideas for earlier deliveries. The reason we could use this program to such a varied degree across the entire semester was mainly because we had a member who felt confident using it. The disadvantage that followed from this is that only this person could edit, create and share this animation. When this person then became temporarily unavailable, the rest of the group was left without the capability of doing anything with the demo.

Collaboration in student communities

Collaboration challenges

One of the biggest challenges within student communities is the asynchronous nature of the work done. As students, the responsibilities and lectures we have can vary wildly from student to student. This makes it especially challenging to find a time slot where every group member can collaborate effectively. This is especially critical when decisions about the project need to be made, so we made effective use of communication channels such as Slack outside of meeting hours to make decisions that need input from all group members.

The most important part of collaboration is communication, it is essential for the very existence of collaboration. If we had no way of communicating, there would be no collaboration, as each student would only be able to work alone. It is therefore extremely important that we as students are able to communicate effectively with one another. Interacting with your fellow students and collaborators is a skill that is essential to any collaborative process, which means that the better communicators your team members are, the easier the process will be. You will also experience higher quality in results with team members who are good at communicating.

Awareness is another important aspect of collaboration and communication, and it affects several aspects of the collaborative process. Collaboration is at its core something that involves

many different individuals with their own thoughts and internal motivations. Because of this, you have to be aware of your collaborators and consider their thoughts and opinions. This is also especially relevant it comes to coordinating work across team members, so no two members are trying to solve the same problem independently.

Settling on the “best” idea in a creative process can be challenging, and in our group there has been a massive output of creative ideas and possibilities for design. It has been hard for the group to end up with one well-defined basic idea, as there has been continuous suggestions towards the improvement of the design.

Challenges in knowledge sharing

Knowledge sharing is extremely important in a project such as this, since the individual pieces of the project are all interdependent. Our group has primarily met in person, which has made the sharing of knowledge and experience easier, but in contrast it has been hard to get work done if one or more team members were absent, as knowledge sharing is markedly harder to do effectively over text.

Knowledge sharing is a form of collaboration, so it follows that communication is important, but it’s also worth noting that awareness is an integral part of the sharing of knowledge. When knowledge is shared amongst several people, you need to be aware of the others, are you perceiving what they are trying to communicate, and are they understanding what you are trying to get across? This overlaps a lot with information sharing, this is because it is impossible to transfer knowledge directly. Knowledge is not explicit and is relative to the person that has it, knowledge sharing has to be done through sharing of information, and then helping the receiving parties with transforming the information into usable knowledge.

Another challenge related to knowledge sharing is the aspect of experience with the subject matter. Students are not trained pedagogues and have little experience in teaching others, in addition, it might be that the student has only rudimentary knowledge of the subject at hand himself. All this compounds into the fact that teaching other students is hard to do, and especially in subjects that require a lot of experience.

Some subjects are harder to teach than others, and our graphical work has been done by a team member in a program only he has access to. This means that he’s solely responsible for all work done on prototyping the like. The consequence of this is that this specific knowledge won’t be shared as easily with the rest of the group.

Game evaluation

The design of the collaboration for this game needed to remain rooted in the fact that it was about IT development. The most common way to collaborate in software development is in verbal communicating in a meeting, and considering how effective and natural this way is, we

decided to choose this as the main way of communication. The game should also try to maximize discussion between the players and provide much support in-game to facilitate discussion. This is something we feel is accomplished well: The players have to cooperate by together planning out the level ahead of time. They will also be granted different visuals and abilities, both of which are essential to completing the task. This will ensure that the players have a lot to say to each other. Naturally, verbal communication has some drawbacks: It has a lot of dependence on the will and social ability on the players to establish a proper conversation. We're also used to be able to talk with full sight of each other, and being able to gesture to each other when words are not enough to convey or meaning. Since we force the player to not see each other's screens, they could lose a lot of this ability to uses visual signalling among themselves. Thankfully, we have some extra dynamics to support verbal communication: The "pings" on the map, as well as the visualization of instructions on the environment (both explained above), allows much richer communication, covering the pitfalls that would otherwise be present with talk-only.

A general drawback still exists in that we place a lot of trust in how the game teaches the players to properly coordinate their actions: Not only must the tutorial and levels facilitate forming good information flow, but make them understand the game concepts of programming, robot abilities, and the task on each level. Naturally, we can try as much as we can right now to make sure the game is intuitive. When designing the dynamics and demo, we did just that by picturing which metaphors or concepts are easiest to understand, and where the game could create complications with the user's understanding. This revealed another, inherent drawback of the game: The game is a natural hotspot for misunderstandings or confusion when solving a level. This is because it's a puzzle game set in a "real" world, and requiring the correct interaction with and around objects such as quest-givers, animals, buttons and trees. Any player could be convinced of some characteristic or behavior in these objects that goes against our implementation. But in terms of collaboration, one could view this as a positive point. If a player misunderstands something, this can lead to some discussion and reflection among the players, since it is a natural obstacle to the completion of the task. They may then experience and learn something new about how collaboration also requires a shared understanding of the environment.

Sources and knowledge evaluation

A lot of our design is grounded in the book Learning With Digital Games (Nicola Whitton, 2009)[1], particularly chapter 6, which encompasses "Designing a digital game for learning". We have also looked at "The Gamification of Learning and Instruction Fieldbook: Ideas into Practice"[2] for inspiration on the design process. We have also based our design upon the lectures we've received in the course, and we've been using the lecture notes as supporting literature.
We've also acquired knowledge through some supporting literature: What we've read of "Designing a digital game for learning" has been very useful. But by far the most useful source to plan and design our game with has been the slides in this course. Especially those slides that

describe the collaboration mechanisms has been useful in designing a game that supports the players in learning IT development as a collaborative process in the best way possible.

Personal Competencies

The various members of the group have had a number of different competencies that have helped during the planning and execution of the project.

Prototyping

Several group members have done prototyping in other courses, and have prior experience in creating graphical user interfaces and concept designs, which is useful for the creation of a good prototype.

Software development

Most of the group members are somewhat experienced in software development, and have done the type of development that would be required for the kind of game we're designing. This gives us unique insight into how the product could be completed after finalizing the prototype. It has also assisted us in planning the work itself.

Game development

Amongst the group, several members are taking the "Interaction Design, Game and Learning Technologies" specialization and have experience in the kind of development process we're doing for TDT4245. Classes like "Cognitive Architecture" have been useful in understanding cognitive limitations in understanding new game concepts. One member of the group is a professional game developer; working as a programmer on a strategy game. Another member joined the GameDev group at NTNU Hackerspace fall 2015, currently a mentor for development of the new Virtual Reality (VR) game and participate in Game Development meetups hosted by Abakus GameDev.
Previous experience with game development has eased the work with this project, and helped ensure a good result.

Graphical design

Several members of our group have extensive experience with graphic design through a wide variety of projects requiring the use graphical software, gained via both their studies and their hobbies. This has been useful when creating mockups and an overarching art style for the game.

The rest of the group have at least some experience with graphic design from subjects like "Human-Computer Interaction", "Graphics and Visualization", "User Interface Design" and other

interaction subjects. One member has learned about both heuristic evaluation and data-driven design through a class at UC Berkeley titled “User Interface Design and Development”. This has made it easier to collectively work on and understand visual illustrations for mocking up the game concept.

Teamwork

All members of the team have priorly participated in a number of student projects, which has helped instill good work ethic and solid procedures for teamwork. Most of the group has also either completed or is currently undergoing the subject “Experts in Teamwork”, which is intended to instill awareness of what creates an effective team, and what each member can do to ensure the team works well. This has helped the team work efficiently together throughout the project.

Discussion and Conclusion

Discussion

Traditionally, learning has been achieved in a classroom environment through a one to many distribution. As the internet have since the 90s become widespread and it is now commonplace to have access to several devices that connect to the internet, a world of opportunities for learning has opened up. Learning can also be a collaborative process, which require participants to share information, discuss alternative options and coordinate activities to realize an agreed upon solution.

Collaborative processes can also be gamified, as has been attempted in our game concept: netCademy Collaboration Games (www.netCademy.com). Our prototype show an early implementation of a game meant to teach teenagers an aspect of IT development through a collaborative process.

Learning by Doing can be achieved through netCademy Collaboration Games through the encouragement of modifying the game. Tynker (www.Tynker.com), a creative computing platform where millions of kids have learned to program and build games, makes Minecraft modding easy (www.Tynker.com/minecraft/). By allowing netCademy players to contribute to the development of new levels, much in the way that Wikipedia works, students can find new sources of motivation for continued learning.

Conclusion

The game concept netCademy Collaboration Games (www.netCademy.com) enables learning through collaboration, so that players need to share information, communicate and coordinate to solve challenges.

Our example level “Save the cat” gives an introduction to the wide variety of levels that can be developed, which enable for further communication, sharing and coordination as players upon completing the game can be invited to ideate new levels, contribute artwork, sketches and collaborate towards the design of new and engaging levels.

Kids and teenagers already spend countless hours modifying and building Minecraft maps, experience the joy of creation. This user group is therefore expected to participate in the development of netCademy levels if given the possibility, so that they can invite friends and family to play their levels.

Linden Research launched Second Life in 2003, the pioneering virtual world that’s been enjoyed by millions of people and seen billions of dollars transacted among users in its economy. In 2013 their portfolio were expanded with Blocksworld, a lighthearted build-and-play system on the iPad for kids and grownups alike. Users of Blocksworld have already generated more than 4 million creations, proving the popularity of these services.

netCademy Collaboration Games have obvious elements for enabling reinforcing learning, considering that players can take an active role in developing the eco-system - which further help achieving the goal of teaching teenagers IT development as a collaborative process.

Sources

[1] **Learning with digital games: a practical guide to engaging students in higher education** - Nicola Whitton - Routledge - 2009
[2] **The gamification of learning and instruction fieldbook: ideas into practice** - Karl Kapp, Lucas Blair, Rich Mesch - Wiley - 2014
[3] **Interactive tabletops in education -** P. Dillenbourg, M. Evans - 2011
[4] **Multi-touch tables and the relationship with collaborative classroom pedagogies: a synthetic review** - S.E. Higgins, E.M. Mercier, E. Burd, A. Hatch - 2011
[5] **Part II: Teachers' Own Use of the Internet and Mobile Tools** - Kristen Purcell, Alan Heaps, Judy Buchanan and Linda Friedrich - 2013 - Retrieved from: http://www.pewinternet.org/2013/02/28/part-ii-teachers-own-use-of-the-internet-and-mobile-tools [downloaded 07.04.2017]