UPPSALA
UNIVERSITET

UPTEC STS 17018
Examensarbete 30 hp
Juni 2017

From Intent to Code

Using Natural Language Processing

Adam Byström

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

From Intent to Code using Natural Language Processing

Adam Byström

Programming and the possibility to express one's intent to a machine is becoming a very important skill in our digitalizing society. Today, instructing a machine, such as a computer to perform actions is done through programming. What if this could be done with human language? This thesis examines how new technologies and methods in the form of Natural Language Processing can be used to make programming more accessible by translating intent expressed in natural language into code that a computer can execute. Related research has studied using natural language as a programming language and using natural language to instruct robots. These studies have shown promising results but are hindered by strict syntaxes, limited domains and inability to handle ambiguity. Studies have also been made using Natural Language Processing to analyse source code, turning code into natural language. This thesis has the reversed approach. By utilizing Natural Language Processing techniques, an intent can be translated into code containing concepts such as sequential execution, loops and conditional statements. In this study, a system for converting intent, expressed in English sentences, into code is developed. To analyse this approach to programming, an evaluation framework is developed, evaluating the system during the development process as well as usage of the final system. The results show that this way of programming might have potential but conclude that the Natural Language Processing models still have too low accuracy. Further research is required to increase this accuracy to further assess the potential of this way of programming.

Handledare: Magnus Lundstedt
Ämnesgranskare: Joachim Parrow
Examinator: Elísabet Andrésdóttir
ISSN: 1650-8319, UPTEC STS17 018

Populärvetenskaplig sammanfattning

Programmering och förståelsen för hur man interagerar med en dator blir allt viktigare i vårt alltmer digitaliserade samhälle. Programmering blir ett verktyg för att ta till sig och kontrollera den nya tekniken. Detta skapar ett stort behov av personer med denna specifika kompetens men också frågeställningar om tekniken verkligen är tillgänglig för alla.

Ny teknik kan vara lösningen på detta. Idag har de flesta människor en dator i fickan och behöver bara säga “Siri” eller “Okej Google” för att denna dator ska kunna göra allt från att svara på deras frågor till att skicka ett SMS eller boka en restaurang. Det har under senare år gjorts stora framsteg inom maskininlärning; att en dator kan lära sig saker genom att kolla på stora mängder data och Natural Language Processing; en dators förmåga att förstå mänskligt språk. Denna teknik gör det möjligt att få en dator att tolka en människas avsikt med högre säkerhet och flexibilitet än tidigare.

Denna studie undersöker möjligheten att använda nya tekniker inom Natural Language Processing som ett hjälpmedel att skapa förståelse för hur man interagerar med en dator och som ett verktyg att lära sig programmering. Under studien utvecklas ett system som omvandlar användarens avsikt, uttryckt i mänskligt språk, till kodspråk som sedan kan exekveras av datorn. Detta system implementeras i ett spel där användaren beskriver vad hen önskar ska hända i spelet, i form av spelpjäser som utför handlingar. Denna vilja översätts därefter till kod som visas för användaren och sedan exekveras för att styra spelet.

För att utvärdera dessa teknikers framtida möjlighet att användas för att översätta mänsklig avsikt till kod utvecklas ett testramverk. Ramverket utvärderar såväl teknikens begränsningar som användarupplevelsen av att programmera genom att beskriva sin avsikt i mänskligt språk.

Resultaten av studien pekar på att denna typ av programmering potentiellt skulle kunna bidra till att föra mänskligt språk och programmering närmare varandra. Studien visar dock att det krävs vidare forskning och utveckling inom Natural Language Processing för att öka noggrannheten av modellerna. Det krävs också vidare utveckling för att kunna modellera ytterligare delar av kontexten av användarens avsikt. I dagsläget håller dessa modeller låg noggrannhet. Detta medför att framförallt personer utan, eller med begränsad, programmeringserfarenhet har svårt att använda systemet. Med ökad noggrannhet hos dessa modeller skulle system, likt det som utvecklas i denna studie, på ett mer korrekt sätt kunna representera användarens avsikt, oavsett användarens tidigare programmeringserfarenhet. Detta skulle bana väg för mer forskning för att vidare undersöka dessa systems påverkan på förståelse för programmering och hur en dator fungerar.

Acknowledgements

I would first like to thank my supervisor Magnus Lundstedt and everyone else at Precisit for great brainstorming sessions and great support during my work with this thesis. It has been a very pleasant experience to work alongside you all.

I would also like to send much gratitude to my subject reader Joachim Parrow for welcomed guidance and feedback. Starting off with a crazy idea, you helped me realize it.

Others who helped realize this thesis and deserve boundless appreciation are the people who took time to participate in this study; Oscar, Elisabeth, Selma, Arvid, David and Magnus. Thank you all!

And finally, a huge thank you to Linn Öfverstedt for being my STS partner in crime, making the most out of our education by making it our own.

1. Introduction

With a society that is becoming increasingly integrated with IT and technology, the need to understand and interface with computers is becoming more and more important. With programming being a core skill for doing this, the need for programmers will only increase. According to Code.org, a non-profit organisation dedicated to expanding access to computer science and programming, in 2020 there will be 1.4 million computing jobs in the United States alone, but only 400 000 computer science students to fill them (Code.org, 2013). In a newly published report on the technology outlook for Nordic schools, one of the key trends observed was an increased importance of coding and Computational Thinking: "the skills required to learn coding combine deep computer science knowledge with creativity and problem-solving" (Adams Becker et al, 2017). According to the report, many make the case that coding should be embedded as a part of primary and secondary education curricula, something that is now done in Finland (Yle, 2015) and will be done in Sweden, starting from 2018 (Regeringskansliet, 2017).

Parallel to this, a lot of progress has been made in the field of Natural Language Processing, the possibility for a computer to learn, understand and produce human language. This progress is primarily made possible by four different factors; increased computing resources, increased availability of data, improved machine learning methods and finally; improvements in the field of linguistics (Hirschberg & Manning, 2015). These improvements have made it possible for computers to, with a much higher accuracy than before, understand human language and humanity over all. With virtual assistants like Apple's Siri, Cortana and Google Assistant, we humans can interact with a computer in a whole new way, asking it to handle interactions with our smartphone apps or answering questions. What if this technology could be used to develop new ways of interacting with a computer? What if it could replace programming, making technology accessible for everyone?

This thesis studies how a computer can, using the available tools for Natural Language Processing, translate an intent, expressed in human language, into code, making Computational Thinking and programming more accessible.

To do this, two different approaches are developed based on Part-of-Speech tagging and either Dependency Parsing or Semantic Role Labelling. As a testing framework, a system that takes an intent, expressed in natural language, applies one of the approaches and generates Actions, Objects and their relations, is developed. These are then semantically matched using the lexical resource WordNet to handle ambiguity. Finally, code is generated that is executed to control a game environment. To evaluate the two approaches, an evaluation framework consisting of two parts is developed. The first part evaluates the technical challenges of implementing the approaches and their limitations. The second part is aimed at evaluating the user experience of the system and its

potential for giving a greater understanding of programming and Computational Thinking.

This study can conclude that this way of programming might have potential, especially using Semantic Role Labelling. But due to the low accuracy and flexibility of the Natural Language Processing models, it is difficult to evaluate its future potential. This study shows that the errors that occur in the Natural Language Processing models creates confusion and frustration to a degree where the system's purpose becomes almost impossible to evaluate. Further research and development of Natural Language Processing models with higher accuracy is required to be able to further research similar approaches to programming as studied in this thesis.

2. Background

As a basis for this thesis, the background covers three main areas of research. First, to analyse the underlying reasoning involved in programming, Computational Thinking is explored; what it is and how abstractions help with solving problems in a structured way. Second, the thesis gives a brief overview of alternative methods of programming. The overall goal of this thesis is to make learning programming more accessible. This background section explores many attempts to do so, with everything from games to Natural Language Programming, and the lessons learned from that. Third, a range of Natural Language Processing techniques are explored, with the goal of extracting the intent from natural language sentences; Part-of-Speech tagging, Word-Sense Disambiguation and Similarity, Dependency Parsing and Semantic Role Labelling.

2.1 Computational Thinking

Computational Thinking is using the concept of abstraction to solve problems, design systems and understand human nature in a way that is derived from computing. Wing (2008) describes the abstractions of Computational Thinking as richer and more complex than the ones found in other fields, such as mathematics or physical sciences. They are often abstractions "beyond the physical dimensions of time and space", but at the same time being limited by them (Wing, 2008). Wing (2008) also predicts that this way of thinking will be a central part of everything in the future and that this introduces new educational challenges; how and when should people learn Computational Thinking?

One basis for Computational Thinking is the selection process of which details to highlight and which to hide with abstraction (Wing, 2008). Another is the concept of working with several layers at the same time, with standardised connections between them. Examples of this in computing are for example the network stack or different components in a larger system that interfaces with API calls, making it possible to interact with other layers without deeper knowledge of them (Wing, 2008).

From the perspective of Computational Thinking, a computer program is a list of step-by-step instructions that tell the computer what to do in a very precise manner. Sáez-López et al (2015) wrote that the creation of such a computer program does not require a special expertise, just a structured way of thinking. It all boils down to how a computer, either a computer in the classical sense or a computing human, can solve a problem by choosing the right abstractions and computer for the task at hand (Wing, 2008). According to, among others, Wing (2008) and Sáez-López et al (2015); to ensure a broad understanding and use of Computational Thinking, as needed in the digitalizing society, Computational Thinking should be taught to everyone in the early years of childhood.

2.2 Alternative means of programming

To make it easier for children as well as adults to learn and practise Computational Thinking and programming, alternative means of programming have been developed. Many of them use abstractions for the underlying computer instructions, such as graphical elements or natural language.

2.2.1 Learning programming

Since the early 1960's, there has been a substantial amount of research in developing tools to make it easier for people to learn programming. New programming languages and environments have been developed. They focus on different aspects of programming and how these can be learnt: how to structure a solution to a problem and how to write unnatural syntax and commands (Kelleher & Pausch, 2005).

Kelleher and Pausch (2005) divide the different aspects of programming into two different categories based on what they conclude are the biggest obstacles in learning programming. These are expressing the intention of the program to the computer and understanding how the computer executes this intention in the form of instructions. Several languages have been developed to make it easier for a user to express intentions in a syntax that the computer can understand. According to Kelleher and Pausch (2005), novice programmers often have problem with translating intention to code. Programming languages in this category primarily focus on either making the syntaxes easier to learn or by using alternative ways in which a user can express intention to the computer.

Several different approaches to making the syntax easier have been taken. These include: simplifying the language, limiting the domain of problems to be solved by programming and preventing syntax errors. As Kelleher and Pausch (2005) conclude: many general-purpose languages use syntaxes and names of commands that feel unfamiliar to users since they originate from the computer rather from the human language. Languages like BASIC, Blue and Junior Java tackle this by adopting the programing languages vocabulary to English, as well as deriving syntaxes and concepts from everyday life. By doing this, the scope of solvable problems is limited, as

described by Wing (2008), but this makes the language look and function like a traditional programming language. This in turn, according to Kelleher and Pausch (2005), makes the transition to a general-purpose language easier.

To prevent syntax errors, the most common method is to use a programming synthesizer with a finite set of predefined building blocks or templates with blank sections with space for a specific statement, condition or phrase. By limiting the combinations of these templates, syntactic errors can be avoided (Kelleher & Pausch, 2005). This concept is also used in programming languages that focus on expressing intention in alternative ways to the computer.

Several ways to express intention to the computer have been developed with great success. One approach in which programming languages try to abstract out the syntaxes is using user actions, such as button presses in a game, within a digital environment, to define a program. Another approach is to create objects that in some sense represent units of code that can be moved around and combined in different ways (Kelleher & Pausch, 2005). These objects can be in the form of graphical elements on a computer screen but also in the form of physical building blocks, such as in Electronic Blocks from Wyeth and Purchase (2000). Environments for learning programming especially targeted at children have increased in popularity the last couple of years. Some of the most popular of these are graphical, object based programming languages like Alice, Scratch and the website Code.org, and where the last two are based on the block-based programming language Google Blockly (Good, 2011; Kalelioğlu, 2015).

Scratch is a block based programming language created by the Lifelong Kindergarten group at the MIT Media Lab, as an extension on Google Blockly. The Scratch-blocks, that focus on creating interactive stories, games and simulations fall into seven different categories. These are: motion, looks, sound, pen, control, sensing operators and variables (Sáez-López et al, 2015; Lifelong Kindergarten Group, 2017). Results from studies performed by Sáez-López et al. (2015) show that, because of the playfulness and the graphical nature of the language, coding in this interface is much easier than traditional general-purpose programming languages. Brennan and Resnick (2012) have developed a framework of concepts used in many programming languages and that are also implemented in Scratch using different sets of blocks. These are Sequences, Loops, Events, Parallelism, Conditionals, Operators and Data.

Sequences are described as dividing an activity or task into a series of smaller steps or instructions that each do one thing. **Loops** are described as executing several of these instructions several times without repeating the instructions themselves. **Events** are "one thing causing another thing to happen" (Brennan & Resnick, 2012); for example, when a button on the keyboard is pressed or when an object on screen is clicked. **Parallelism** is when several sets of instructions are executed simultaneously, or in parallel. **Conditionals** are the concept of making decisions based on if a condition is fulfilled or not**. Operators** are instructions that perform numerical or string manipulations using mathematical, logical and string expressions, or operations. Lastly,

the **Data** concept involves storing, retrieving and updating values, and which in Scratch are represented by variables and lists. Variables can hold one value, a number or a string, while lists can hold a collection of numbers and strings.

Studies on how games can be used in education and to teach programming show that teaching concepts of programming and Computational Thinking through a game can make it easier to learn as well as be more fun and engaging (Bromwich, Masoodian & Rogers, 2012). Bromwich, Masoodian & Rogers (2012) conducted a study where they developed a game for learning and practising the basic concepts of programming in an engaging way, without using traditional syntax. They saw that students who learn programming traditionally are taught syntax first and then rushed into more complex projects without practising the basic concepts, like loops. Their game environment consisted of a 2D word with a visual programming editor where commands, conditional statements and loops were represented by circles with text. These circles are then connected to control an avatar. The goal was to, for each level in the game, navigate it through a maze with an increasing level of complexity. The study showed that a game environment is both a fun and inspiring way of learning and practicing fundamental programming concepts.

2.2.2 Natural Language Programming

What if we could use human language, or "natural language", for programming? Could that make it easier for novice programmers to get into coding? Up to date, several attempts to create such a programming language have been made. Languages such as HyperTalk, Cobol or Inform 7 all are based on this notion, writing code that is as close to the human language as possible, making programming more accessible. But there have also been other attempts with less noble intent. LOLCode is a programming language that its creators call "An esoteric programming language" (LOLCode, 2017). It is based around internet slang and so called memes, a set of culturally significant references, primarily the internet's obsession with cats.

Several studies into the use of natural language as means of programming, both in the context of code generation, but also for comprehension, debugging and collaboration have also been made. As early as 1966, Sammet (1966) was discussing the potential of using the human language as a programming language. Sammet (1966) describe that the challenge of using English or any other human language for programming will always be for the computer to accurately resolve any ambiguity. For example, by querying the user. It is also important that the computer can determine the correct interpretation of a sentence where a number of possible syntactic interpretations could be made. She speculates that there could be patterns to these ambiguities and syntactic variations that the computer potentially could learn and thereby reduce the amount of query-interaction with the user (Sammet, 1966).

Empirical studies on the feasibility of programming in natural language have, over the years, yielded varying results. Biermann et al (1983) showed promising results in their

study using the Natural Language Programming system NLC on a limited domain of operations on data tables and matrixes. As concluded in the study, students in a first course in programming could quickly learn and get started programming with the subset of the English language implemented in the system. They also concluded that the vagueness and ambiguity of natural language did not significantly affect performance when used with the limited domain of NLC (Biermann et al, 1983). Capindale and Crawford (1990), in another study, found that another natural language system, Intellect, could be used successfully in limited querying of databases in the case when the stored data is known to the user. Although successful, this study found that one of most potent limitations to the system was its inability to handle context and grammatical variations as well as the systems limited vocabulary and functionality (Capindale and Crawford, 1990). Another, more sceptical approach comes from Miller (1978). In his study, he found that descriptions of programs from the users are often incomplete in relation to the code of the program. His study also showed that the users' descriptions state the actions first and then the conditions in which the action is performed, and which is the opposite to how a computer handles conditions (Miller, 1978).

Among the more recent publications on the subject, Good (2016) performed several studies on game development as a basis for Natural Language Programming. First Inform 7, a programming language primarily focused on developing interactive stories with a 'read like English'-syntax was studied (Good, 2016). The goal of this study was to identify potential problems that arose when adults with no prior programming experience were faced with the programming language. Two major groups of problems were discovered. First, the differentiation between what natural language should be interpreted as programming language and what should be interpreted as strings. Second, problems related to using natural language as a programming language in general, such as the use of synonyms of syntactic keywords or the use of incorrect Inform 7 syntax (Good, 2016).

In the second study, Good analysed how children, aged 11 – 12 years, would naturally describe events and behaviours in a game by letting them play a game that embodied one of several programming elements (conditions, Booleans etc.). The results of this study showed poor performance for code generation, primarily because of problems with incomplete descriptions, much like the findings of Miller (1978). A third study found that the children showed great improvement when using laminated cards with different programming elements that should be paired together in a non-digital setting.

Good (2016) could confirm the findings of Miller (1978), that incomplete statements result in a large portion of encountered errors. Good (2016) also confirmed the ambiguity discussed by Sammet (1966) and her own findings regarding strings in contrast to natural language "code". Based on these findings, Good (2016) developed seven design principles for developing a Natural Language Programming language. They are divided into two categories; the first one code generation and the second is comprehension, debugging and collaboration.

Code generation

1) **Constraint expression during program generation.** Good suggests that, in order to prevent novice programmers from running into syntax errors, the domain of expressions should be limited. This could also be combined with an autocomplete feature in the case of a text-based language.
2) **Clearly distinguish 'code' from free-text.** When using natural language as a programming language it should be clear to the user when natural language should be interpreted as 'code' and when it should be interpreted as strings.
3) **Highlight distinctions between different computational categories.** It is, according to Good, important to differentiate between different programming constructs, such as states and actions, so that they don't get mixed up.
4) **Make underlying structure visible to avoid errors of omission or commission.** It should be clear how different programming constructs go together, and, when there are missing constructs, what they are.

Comprehension, debugging and collaboration

5) **Provide a full-sentence natural language description of the code.** Good found in her studies that it was vital that the user easily could understand and review the code they have just written. She suggests that a description of the code in natural language could solve this.
6) **Use 'natural' natural language.** The studies also found that full-sentence, everyday language should, as far as possible, be used for both syntax and in error messages to the user.
7) **Do not suggest the system can engage in dialogue when it cannot.** In one of the studies, using the programing language Inform 7, the error messages were verbose and "pseudo-conversational". In the study, Good found that this was confusing rather than helpful and so suggests to not portray the system as more capable than it is.

2.3 Natural Language Processing

Human language, or natural language, is complex and ever evolving. One of the first recognized attempts to, in the tradition of scientific theory, create theorems and rules for understanding and generating language was made by Chomsky (1957). He found that grammatical structure, or Grammatical sequences and Ungrammatical sequences, could be described in terms of logical rules, as opposed to the semantic meaningfulness of the sequence. The classical example of this is the sequence "colourless green ideas sleep

furiously" (Chomsky, 1957) that does not hold any valid semantic meaning but, according to Chomsky (1957), is still grammatically valid.

Even though there have been debate and criticism of these theories, Chomsky has been renowned for the scientific approach he brought to linguistics (Sampson, 1980; Markus, 1995). This scientific approach has made a great impact in the field of Natural Language Processing, NLP. NLP is the technique for making a computer understand natural language. The Natural Language Processing community has grown since the 1960s and has focused on a set of tasks. Some of these are Machine Translation, Named Entity Recognition, Part-of-Speech Tagging, Parsing, Question Answering, Relationship Extraction, Speech Recognition and Word Sense Disambiguation. In the following sections some of these techniques, related to this thesis, are explained in more detail.

2.3.1 Part-of-Speech Tagging

Part-of-Speech tagging is the process of labelling a word in a sentence with what part of language it belongs to, such as verbs, adverbs and nouns. A lot of research and work in general has been done in this area of Natural Language Processing. This is mostly thanks to the development of large corpora, structured set of texts (Martinez, 2012).

The two major challenges for Part-of-Speech tagging are ambiguous words; words that, depending on context, are part of different parts of speech and words that are unknown. Since the Part-of-Speech taggers are trained on a limited set of corpora (a finite amount of text), if that word does not occur in the training data, it is harder for the tagger to label the word correctly (Martinez, 2012). Several methods for Part-of-Speech tagging and trying to address these problems have been made using different approaches. These can be divided in two categories; rule-based and probabilistic methods.

Rule-Based methods use, as the name suggests, rules, in the form of rule sets of allowed sequences of tags. In most cases these are created manually by experts in linguistics, something that Martinez (2012) describes as "too inefficient to be practical". Due to this inefficiency, Brill (1995) developed a method, where a model can learn rules from corpora using Transformation-based learning, as depicted in *Figure 1*.

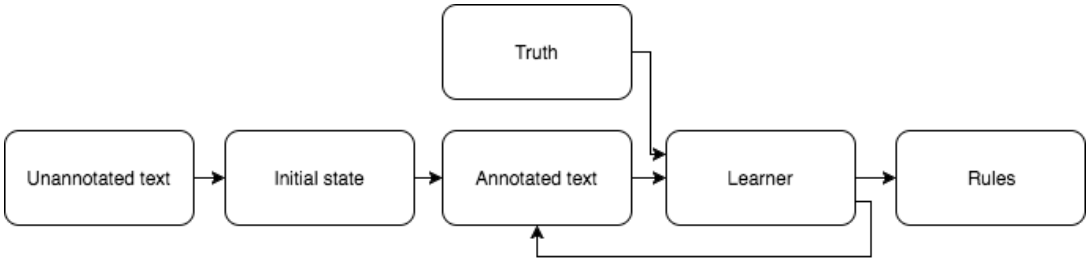

Figure 1: Transformation-based learning, (Brill, 1995)

The method starts off with an unannotated set of texts that it gives an initial annotation. This is then processed by a learner that compares it with a pre-labelled text, referred to as truth. By doing this iteratively the learner automatically creates new set of rules (Brill, 1995).

In the early 1990s, a different approach to Part-of-Speech tagging started replacing the rule-based methods with probabilistic methods, and primarily Markov model taggers. The Markov models, in combination with increased access to structured data, transformed the whole field of Natural Language Processing altogether (Martinez, 2012). By not needing to write rules that tended to be extremely complex and often had a low amount of flexibility they could reach a higher effectiveness with the same or higher accuracy (Markus, 1995).

Modern Part-of-Speech taggers are often based on Hidden Markov Models (HMM:s), a variant of the Markov models (Martines, 2012). Markov models, or Markov chains are based on the notion that, for a sequence of random variables that can take on one, out of a finite set of states, a variable is only dependent on the immediately preceding variable, independent of time. In the case of Part-of-Speech (POS) tagging, the POS-tag of a word is only dependent on the POS-tag of the preceding word, independently of where in a sentence these words are.

A Markov chain can also be envisioned as stochastic transitions between states where the transition probabilities to the next state are based on the current state. In a standard Markov model these states are observable and it is therefore possible to compute these transitions. In a HMM the sequence of states is not observable. It is only the output from these states that is visible; the words, in the case of POS-tagging, that can be observed. By training the POS-tagger model on a pre-tagged corpus that is treated as a visible Markov model where the states are observable and probabilities can be computed, it is then possible to apply this model to a new set of words, but as a HMM, observing the word we wish to tag (Manning & Schütze, 1999).

Another probabilistic technique used for Part-of-Speech tagging is the Maximum Entropy method, MaxEnt. In contrast to Markov chains, MaxEnt models assume that the unknown POS-tags are conditionally independent of each other. The MaxEnt model is based on maximizing the entropy of a probability distribution subject to certain constraints. These constraints are based on contextual features observed in the training data, such as number of occurrences of a tag. By doing this, these models have shown to be able to tag words with the correct tag with high accuracy (Ratnaparkhi, 1996).

2.3.2 Word-Sense disambiguation and similarity

One word can have different meanings, fire could for example refer to flames and smoke, to shoot or to terminate employment. The way humans can tell the difference is often from the context the word is used in. Word-Sense Disambiguation (WSD) is a task

in Natural Language Processing that focuses on finding the correct semantic meaning of a word given its context (Navigli, 2009).

At its core, a WSD system works by, given a set of words, using some technique to apply one or several sources of knowledge, for example corpora, dictionaries or other lexical resources, to find the most likely semantic meaning of the analysed word (Navigli, 2009). One of the most used sources of word meaning knowledge is WordNet, a lexical database for the English language, created and maintained at Princeton University (Princeton University, 2010). It uses sets of cognitive synonyms, called Synsets, representing words with approximately the same meaning. Since a word can have different meanings, several Synsets can exist that contain a given word. WordNet also include semantic relationships between words, for example, if one word is a superset of another, and, relationships between adjective antonyms (Princeton University, 2010). The paths formed by these relationships can be used to measure the semantic similarity between two words.

To find the most likely meaning of a word, a set of features is chosen for that word to represent the word's context, for example from Part-of-Speech tagging and Parsing (Navigli, 2009). These features are then used in different ways to classify the word as one of the potential meanings. The two main approaches in which models for this classification are trained are, as is the case for many of the Natural Language Processing tasks; supervised and unsupervised (Navigli, 2009). Supervised approaches use manually pre-labelled data, labelled with their syntactic meaning, to create classifier models. These models are then used to classify new words with syntactic meaning. The unsupervised approaches on the other hand, are not able to classify a word with a specific meaning but rather clusters words with similar meaning, based on them occurring in similar contexts.

2.3.3 Dependency Parsing

Dependency Parsing is a type of parsing based on syntactic dependency grammar. It is based on the notion that the syntactic structure of language consists of words that are linked by dependencies, or "binary, asymmetrical relations" (Nivre, 2010). A dependency consists of a Head and its subordinate words, called the Dependents. These dependents can have different syntactic relationships to the *Head* based on the grammatical context; subject (SBJ), object (OBJ), attribute (ATT) etc. The head and the dependents are often structured as a tree structure where every word has a single syntactic head and each branch is dependent on the word on the top of the branch. Often, the head of the top of the tree is labelled ROOT, so that every real word in the sentence can be assigned to a head.

Within the tree structure, the task of Dependency Parsing becomes mapping the input sentence to one or more tree structures where every word is linked to a head with a dependency label (Nivre, 2010). As discussed by Nivre (2010), there are several approaches to dependency parsing: Context-free Dependency Parsing, Constraint

Dependency Parsing, Graph-based Dependency Parsing and Transition-based Dependency Parsing. An example of a Context-free dependency tree is shown in *Figure 2*.

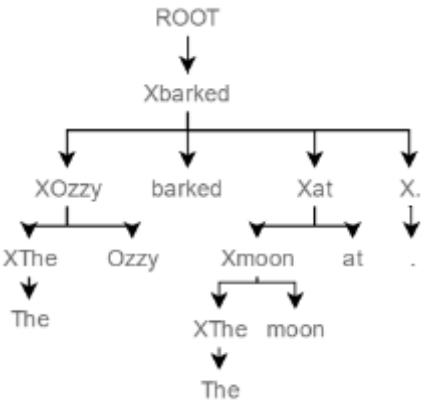

Figure 2: Context-free dependency parsing.

Context-free Dependency Parsing is based on non-terminal nodes, labelled with words, that indicate the top of a subtree, indicated with an "X" in Figure 2, followed by the word label. For example, in the sentence "The Ozzy barked at the moon.", "Xbarked" would be the node dependent of ROOT and would in itself be the head of "barked" as well as the subtrees branching from "XOzzy", "Xat" and "X." (containing the punctuation). This parsing is based on the notion of context-free grammar, CFG, a finite set of rules of binary relations for each non-terminal symbol to a finite string of symbols (Scheinberg, 1960). In their example, a rule could be that a sentence should contain a noun phase and a verb phrase, another that a noun phrase can contain a noun and a determiner. The nodes starting with X would be the different phrases, called non-terminal symbols. Nouns, verbs etc. would be the terminal symbols.

According to Nivre, Context-free Dependency Parsing, holds two important restrictions. Firstly, the grammar is lexicalized, as the non-terminal symbols are indexed by lexical items, or terminal symbols. Secondly, every branch in the tree not connected to ROOT has exactly one word, or a terminal symbol in it. By satisfying these restrictions, it can be classified as a Context-free Dependency Grammar (Nivre, 2010). One issue discussed by Nivre (2010) is the restriction of the Context-free Dependency Parsing, that is limited to strictly projective dependency trees, where each head word represent itself and all the dependents of the word. Another issue is that algorithms for computing this type of lexicalized grammar structures have high complexity; $O(n^3)$, or in worst case $O(n^5)$, with a large set of rules (McDonald et al, 2005a; Nivre, 2010).

Another approach to Dependency Parsing is the Constraint Dependency Parsing, as defined by Maruyama (1990). It is based on a set of Boolean constraints, rather than

binary relationship rules, set on well-formed dependency trees called Constraint Networks. These Constraint Networks then control the branching of the Dependency Tree. Such a Boolean constraint can be, for example, that a noun in singular form must have a determinant. By evaluating every possible dependency tree with these constraints and successively eliminating those where the constraints are violated, when there is only one left, it is known that it is valid (Maruyama, 1990). This makes Constraint Dependency Parsing to be not, in theory, limited to projective dependency trees as is the case with context-free dependency parsing. In practice this is a computationally demanding task, as it is a NP-complete problem and which has, at best, an exponential complexity (Menzel and Schröder, 1998; Nivre, 2010).

The original version of the constraint dependency parsing by Maruyama (1990) was subsequently evolved to take into concern the different importance of these constraints, in order to account for the possibility that no dependency tree was valid (Menzel and Schröder, 1998). As Menzel and Shröder (1998) suggested, instead of giving the assessment function map a constraint of zero or one, give it a weight based on how serious the violation of that constraint is. By then summing up these weights for a dependency tree, the tree with the highest score can be chosen (Menzel and Schröder, 1998). More recent implementation of these concepts have used transformation-based methods, making it possible to have complex constraints other than binary and still maintain efficiency (Foth et al, 2004).

A similar dependency parsing method is Graph-based Dependency Parsing, which also uses scoring of all possible dependency trees for a given sentence. The difference in this method is that it gains its scores, not from specified constraints but rather from stochastic analyses of marked corpora or treebanks, using machine learning (Nivre, 2010). Scores in Graph-based Dependency Parsing, as can be seen with other stochastically based methods, can be calculated in many ways. The fundamental principle for the Graph-based methods score is that it is based on the scores of its subgraphs, most commonly their sum (Nivre, 2010).

A further method that uses machine learning is Transition-Based Dependency Parsing (Nivre, 2010). This method uses a state machine that consists of a set of partial analyses of a sentence, called Configurations and a set of transitions between configurations. There is also a set of terminal configurations, so that when the transition system ends up at one of them, it knows that it is finished. Before finishing, the method applies transitions to move between configurations based on a scoring function. The function scores possible transitions based on a feature vector of the current configuration, where the most important features are attributes of the word, for example, its Part-of-Speech, in relation to its position in the configuration (Bohnet, 2011; Nivre, 2010). By then combining the scores for a complete sentence, it is possible to treat the parsing as a search for the sequence of transition that results in the highest score for the sentence, making it possible to perform in quadratic or linear time (Bohnet, 2011; Nivre, 2010).

Bohnet (2011) showed, when comparing Transition-based and Graph-based Dependency Parsing, that the transition based method could perceive higher level and subcategorization features while the graph based method showed a slightly higher tendency to account for long distance relationships. This was something also noted by McDonald & Nivre (2007) who described it as a trade-off between the graph based, long distance learning of local features and the local learning of global features from the Transition-based Parsing. By combining these models, Nivre & McDonald (2008) managed to improve accuracy for both models, resulting in a significant improvement over previous state of the art models.

2.3.4 Semantic Role Labelling

With powerful Part-of-Speech tagging and Dependency Parsing, as described earlier, it was possible to model the grammatical structure of a body of text, but not to directly analyse "Who did what to Whom and How, When and Where" (Palmer, Gildea & Xue, 2011). It is this particular aspect that is addressed by Semantic Role Labelling.

The concept of Semantic Role Labelling is based on identifying an event and then assigning semantic roles to different words that relate to that event in different ways. By evaluating an event as a verb, surrounded by arguments representing the semantic roles associated with that event, it is possible to model the semantics of the event (Palmer, Gildea & Xue, 2011). The semantic roles can be structured based on specific verbs as a Theta-grid where every verb maps to a set of involved semantic roles that are needed to put the event into a valid context (Palmer, Gildea & Xue, 2011). For example, given the word *give*, a giver, a thing to be given as well as the things final position, would be needed. This would be, using standardized notation, be grouped in a Theta-grid for the word *give*. The standardized notations of semantic roles, also called Thematic roles, found in *Table 1,* as summarized by Saeed (2015) are widely used in semantic role labelling. Using this notation, the Theta-grid for give is [Agent ("the giver"), Theme ("the thing to be given"), Goal ("the things final position")].

Table 1: A set of widely recognized Semantic roles (Saeed, 2015)

Role	Description
Agent	The initiator of some action, capable of acting with volition.
Patient	The entity undergoing the effect of some action, often undergoing some change in state.
Theme	The entity which is moved by an action, or whose location is described.
Experiencer	The entity which is aware of the action or state described by the predicate but which is not in control of the action or state.
Beneficiary	The entity for whose benefit the action was performed.
Instrument	The means by which an action is performed or something comes about.
Location	The place in which something is situated or takes place.
Source	The entity from which something moved, either literally or metaphorically.
Goal	The entity toward which something moves, either literally or metaphorically.
Stimulus	The entity causing an effect (usually psychological) in the Experiencer.

Even though there are some agreement over the existence of these roles, the difficulty of finding out when and where to use them revealed the need for something more than a simple set of semantic roles (Palmer, Gildea and Xue, 2011). One framework by Fillmore (1985) elaborated on these roles by putting them into Frame Semantics. He saw that the assignation of semantic roles was based on a limited set of underlying semantic representations that created a frame for a verb. By specifying these frames, it would be easier to find the associated semantic roles. Based on the theory of Frame Semantics, a lexical resource called FrameNet, and which contains more than 1,200 semantic frames, has been continuously developed since 1997 at the International Computer Science Institute in Berkeley (FrameNet, 2017).

Another widely recognized labelling system is the verb classes developed by Levin (1993). He recognized that the behaviour of a verb with respect to the context and interpretation of its argument was, to a large extent, based on the semantic meaning of the verb. These behaviours were documented in the form of the Levin classes. The Levin classes are a systematic way of labelling verbs based on their existence in pairs of

syntactic frames that closely relate to the meaning of the verb in that particular context. These verbs can then be grouped into classes based on similar meaning and similar syntactic frames. Such a class could be, for example Avoid Verbs (Levin, 1993). Members of this class include avoid, dodge, duck, elude, evade etc. and a syntactic frame, or "Property", could be "We avoided the area".

Additional lexical resources popular in semantic role labelling, described as being created for different purposes, but "surprisingly compatible" by Palmer, Gildea and Xue (2011) are VerbNet and PropBank. VerbNet is the largest online verb lexicon for the English language (VerbNet, 2017). The verbs are classified according to an extension of the Levin classes, with 274 first level classes (VerbNet, 2017) compared with the 240 original Levin classes (Palmer, Gildea & Xue, 2011). Each class is labelled with thematic roles, selectional restrictions on the arguments and syntactic frames with intention.

PropBank, or Proposition Bank, was, in contrast to FrameNet and VerbNet, not developed as a lexical resource but as an annotated corpus to be used for training machine learning models (PropBank, 2017). In later years, it evolved to incorporate semantic roles on a verb by verb basis, where each verb has a numbered set of semantic arguments labelled with non-theory-specific labels; Arg0, Arg1 etc. (Palmer, Gildea & Xue, 2011). According to Palmer, Gildea and Xue (2011) this verb specific approach, with verb specific role labels, has several limitations, namely that it makes it more difficult to compare role labels to define generalizations, which in turn makes it harder to automatically train semantic role labelling models.

In later years, these three lexical resources have been combined in several ways, and initiatives such as SemLink (SemLink, 2013) and Unified Verb Index (PropBank, 2017) take advantage of their combined strengths. Among other things, mapping PropBank annotated instances to relevant VerbNet classes, creating a larger lexical resource to train Semantic Role Labelling models on (Palmer, Gildea & Xue, 2011).

2.4 Related work

Apart from the work done in Natural Language Programming, there has also been some research done that looked at the combination of natural language and code as well as natural language and instruction interpretation. In the case of Natural Language Processing and code, research such as Falleri et al (2010), Kim and Kim (2016), Shepherd, Pollock and Vijay-Shanker (2007), Alsuhaibani et al (2015), Pollock et al (2007), Abebe and Tonella (2010) and Kuhn, Ducasse and Gîrba (2007) has been done on using Natural Language Processing techniques for analysing source code. The focus of most of this research was on extracting names of program elements and concepts using machine learning and different natural language parsers and taggers. Alsuhaibani et al (2015) have the same goal, to analyse the source code but does not use traditional Part-of-Speech taggers that are based on sentence structure but rather the structure of

the code itself, for example tagging a word as a verb if it is found to be the name of a method.

Other research, in the robotics community, has also looked at understanding instructions in natural language. Chen and Mooney (2011) developed a system for relaying navigation instructions to robots based on observations. They used a semantic parser that they trained on Navigation plans that the authors constructed as a set of word state descriptors and a set of action sequences. Other work, for example by Stenmark and Malec (2014), focused on assembly tasks for industrial robots, using a generic semantic parser to create sets of predicate-arguments, based on a piece of natural language. These predicate-argument combinations, or PA:s, are formulated as verbs, being the predicates, and the grammatical arguments according to non-theory-specific labels A0, A1 etc., labelling them as the actor, the theme, the goal or equivalent to that verb.

2.5 Present work

This thesis has the goal to go from a human intent to code and is, in that regard, the inverse of source code analytics. It resembles the work of Stenmark and Malec on industrial robots (2014) but also looks at additional NLP approaches and has the aim of creating a more general approach that is not limited to one single domain.

3. Methods

To evaluate the possibility of using Natural Language Processing as a tool for interpreting intention a set of methodical approaches are developed. This chapter starts off with defining these approaches and the hypotheses that are defined based on them. It then defines this thesis' methodical key concepts and technologies, such as Speech to Text and the game environment used in the user testing in this study. Following that, it describes the performance evaluation and user testing. Lastly, it describes how these methodical approaches have been implemented.

3.1 Methodical Approaches

Two approaches were adopted to evaluate how human intent can be interpreted as code, using the modern tools and techniques in Natural Language Processing (NLP),. Each approach is based on a set of areas of NLP and their available tools as well as a hypothesis about how they will translate human intent to code. The intent, given as individual sentences (s_1,...,s_n), is mapped to a sequence of objects (o_1,...,o_m) and actions (a_1,..., a_k), where each sentence can contain one or several objects and actions. An action a = (c, P) contains parameters P and a condition c. These parameters P can contain one or more of the following parameters: "On object", containing the object that performs the action as well as "how", "with", "target" and "direction", describing

keywords as to how the action should be performed. The two different approaches use different techniques to do the mapping.

3.1.1 Approach 1: Dependency parsing and Part-of-Speech tagging

This approach is based on the idea of Abbot (1983) that a common noun suggests a data type, a proper noun or direct reference suggest an object and that a verb, attribute, predicate or descriptive expression suggest an operator or method. By using Part-of-Speech tagging, different Part-of-Speech tags, such as verbs and nouns can be identified. These tags can then be linked in a Dependency Parsing structure to find how they relate to each other, making it possible to interpret an intent as objects and actions with related properties.

Hypothesis 1: It is possible to, using Grammatical dependencies and Part-of-Speech tags, model a sentence of human intent as a set of objects, methods and their relationships. By structuring these object and methods by their relationships, code can be created represents the user's intent.

The Stanford CoreNLP, a suite of NLP tools, implemented in Python with Natural Language ToolKit (NLTK) is used to test this hypothesis. Stanford CoreNLP uses the log-linear Part-of-Speech tagger written by Toutanova et al (2003) with a Maximum Entropy method that is trained on the Penn Treebank corpora. The suite also includes the transition-based Dependency Parser using Neural Networks, developed by Chen and Manning (2014), also trained on Penn Treebank.

3.1.2 Approach 2: Semantic Role Labelling and Part-of-Speech tagging

Extending on the first approach with the notion that the grammatical structure of the sentence can be used to translate and intent to code, this approach is based on mapping semantic role labels to code structures. By making assumptions that a verb can be translated into an action with different parameters described by the verb's connected semantic roles, it is possible to construct actions and objects based on semantic labels. To further analyse the meaning of these semantic roles, Part-of-Speech tagging is implemented.

Hypothesis 2: It is possible to, with Semantic role labels and Part-of-Speech tags, model a sentence of human intent as a set of objects, methods and their relationships. By structuring these objects and methods by their relationships, code can be created that represent the user's intent.

SENNA, Semantic/syntactic Extraction using a Neural Network Architecture, (Collobert et al, 2011) implemented in PractNLPTools, a Python library over SENNA and Stanford Dependency Extractor (PractNLPTools, 2016) is used to test this hypothesis. SENNA uses a probabilistic Neural network approach described by

Collobert et al (2011) and was trained on the entire English Wikipedia in combination with Reuters' RCV1 dataset (Collobert et al, 2011).

3.1.3 From speech to text

Speech is described as the most important and the most natural way of human communication and for conveying one's intent (Iyanda, Adetunmbi & Obe, 2016). Recently there has been a lot of research focused on gaining higher accuracy in Speech-To-Text conversion and Automatic Speech Recognition using machine learning (Iyanda, Adetunmbi & Obe, 2016). Based on this, several implementations of Voice-To-Text are tested for generating a text representation of the intent, to be used as input to the different approaches described in this report. These included: The Web Speech API (Shires & Wennborg, 2012), IBM Speech to Text (IBM, 2017), Bing Speech API (Microsoft, 2016) and Google Cloud Speech API (Google, 2017).

3.1.4 From code to game

As described by Bromwich, Masoodian and Rogers (2012), a game environment offers a good way for learning and practise programming concepts and Computational Thinking. To leverage this, a game environment where the user expresses intent regarding what they see on the screen and what they want to have happen in the game, is implemented. The game environment also limits the domain that the intent should be mapped to and the command diversity, making it easier to generalise intent patterns. This domain is defined by an environment $E = (EO, EA)$ as a collection of Environment objects $EO = (eo_1,...,eo_n)$ and Environment actions $EA = (ea_1,...,ea_m)$ that contain available objects and permitted actions in the environment.

Studies of earlier systems for using natural language for code generation have shown that a limitation for these systems are their strict syntax with no consideration for word disambiguation (Good, 2016; Capindale and Craford, 1990; Sammet, 1966). By using Natural Language Processing to do Word-Sense Disambiguation and Word Similarity as a less strict mapping method for objects and actions to the environment, a higher accuracy could potentially be reached. For example, if the user expresses an intent as "The boy should move forward five times" in an environment consisting of EO = (character, tree, rock) and EA = (Walk, wave, hit), the system should map "the boy" as being more semantically similar to "character" than to "tree" or "rock" and "go" as being more similar to "walk" than to "wave" or "hit".

3.1.5 Performance evaluation

A framework of different performance measurements is developed to evaluate the different approaches described in sections 3.1.1 to 3.1.4, and hence the possibility of interpreting human intent as code. To be able to account for the ease of development and technical possibilities as well as the ease of use, the framework consists of two

parts. The first part, as described in *Table 2*, evaluates the development process, with continuous testing and observation of technical limitations.

Table 2: Performance measures for development

Property	**Evaluation metric**
Ease of implementation	The perceived ease of use to create an implementation that turns a sentence of human language that expresses an intent, to code.
Amount of generalisation	The perceived amount of domain specific solutions that is needed to be implemented and the possibility to include concepts from the framework of Brennan and Resnick (2012).

The first part is based on how easy it is to implement the different approaches in code, something that is partially affected by the available implementations of parsers, lexical resources etc. Since this gives different conditions for the implementation of the different approaches, making them difficult to compare quantitatively, a personal qualitative evaluation is performed. The goal of this evaluation is measuring the perceived ease of use as well as the correctness, accuracy and the perceived possibility to make generalisations in each implementation. The second part focuses on user testing of the implementation and the impact of Computational Thinking and is described in *Table 3*.

Table 3: Usability properties and evaluation metrics

Property	Description	Evaluation metric
Learnability	The approach should be easy to learn so that it doesn't require lots of training.	Amount of information that is needed to be given to the user beforehand and if additional information is required.
Efficiency	The approach should be efficient resulting in a high productivity from the user.	Time to complete a predefined task using the system.
Memorability	The approach should be easy to remember how to use so that minimal additional training is needed when returning to the system.	*This property will not be considered in this study.*
Errors	The approach should not result in a high amount of errors. Errors that do occur should be easily corrected.	Amount of commands given by the user that can not be interpreted or are misinterpreted by the system and how easy they are to correct.
Satisfaction	The approach should be pleasant for the user to use. The user should like using it.	An interview after using the system to determine the users subjective feelings about the system in general.

It is based on Usefulness as a combination of Usability and Utility, defined by Grudin (1992). Usefulness is the measurement of a system's possibility to achieve a desired goal (Nielsen, 1993). Nielsen (1993) then define usefulness as the overarching structure of the two subcategories; utility and usability, where utility is the question of whether the functionality of the system, in principle, can fulfil its purpose, and usability is the question of how well a user can use that functionality. Usability can then be divided into several properties: Learnability, Efficiency, Memorability, Errors and Satisfaction (Nielsen, 1993). These are evaluated according to *Table 3*. Due to the relative infrequency of use of the system in combination with the limited scope of the project, Memorability is not evaluated.

3.2 User testing

A set of test subjects are chosen to evaluate the usability of the system as well as its relationship to Computational Thinking. These test subjects are selected according to their different levels of programming experience with a minimum requirement that the participants should be fluent in the English language. Even though Computational Thinking is much more than just programming; programming can be seen as a concretisation of Computational Thinking. This makes prior programming experience a reasonably good measurement of the test subjects' initial level of Computational Thinking.

Programming experience is classified into three levels: novice, intermediate and expert. The participants in the study are assigned to these groups based on their own estimate of their programming experience. From each experience level two candidates are selected, one for each Natural Language Processing approach; Dependency Parsing and Semantic Role Labelling. Each participant is first given instructions on how to interact with the implementations and what objects and actions that are implemented in the environment. They are then instructed to complete a task consisting of moving the character object to the goal object in the game. To complete this task, the character additionally must traverse an obstacle; a river, where the only river crossing is blocked by a tree.

During this task, evaluation metrics are recorded in accordance with *Table 3*. On completion of the task, the participants are invited to further explore the system for a short period of time. Data on every interaction with the system is automatically collected in digital log-files; logging the user input, the labelling done by the NLP models and the generated code. After the user has used the system, they are asked a series of questions to determine their feelings on using the system.

3.3 Limitations

Due to the large number of tools, models, lexical resources and tagged corpus exclusively related to the English language, this thesis limits its scope to translating intent expressed in English. This said, it is fair to assume that, given enough data and time training models on that data, the same or similar techniques to those used in this thesis could be used on other languages. A limitation in all languages, English included, that affects the accuracy of the approaches in this thesis, is the low number of imperatives in the data that these tools and models have been trained on. Most Natural Language Processing tools and models have traditionally been developed and evaluated on news articles and other descriptive texts that contains a low number of imperative sentences.

Based on the goal of this thesis, to evaluate the described approaches in the context of novices learning Computational Thinking, the domain of programming concepts is limited to the framework of concepts described by Brennan and Resnick (2012). Because of the difficulties in separating strings from instructions in natural language as

described by Good (2016), strings are not considered in this thesis and are left for future work.

3.4 Implementation

3.4.1 Testing interface

The described approaches and methods are implemented in a web interface with a predetermined environment of available actions and objects as described in *Table 4*.

Table 4: Testing environment

Actions and parameters	Objects
▪ Walk (*direction* / *target*) ▪ Jump (*how*) ▪ Cut (*target*, *with*) ▪ Eat (*target*)	▪ Character ▪ Tree ▪ Axe ▪ Cow ▪ Goal

The web interface, as shown in *Figure 3*, is divided into three sections; A, B and C.

Figure 3: Web interface.

In section A, the user is given instructions of how to use the system. In section B, a code editor is inserted where the code generated by the system is shown. The code editor is also equipped with controls for executing the input code. The last section, section C, holds a simple game where objects could move around in a grid, performing actions on each other.

3.5 Speech to Text Implementation

The first solution tested for translating voice to text was is the Web Speech API, a JavaScript library built into modern web browsers. Its initial testing yielded poor results with low accuracy and this solution was therefore abandoned. Following Web Speech API, several commercially available voice-to-text solutions were tested using their respective demo versions. In these initial tests, Google Cloud Speech API Beta showed the most promising results and was selected for further testing and implementation. Further testing showed good results but due to the time constraints and the fact that this solution implements new and not widely adapted technologies and standards, there was not enough time to implement it in the web interface.

3.6 Study 1: Dependency Parsing

The Dependency parsing implementation is based on the Stanford Dependency Parser which takes a string of text and, in this implementation, returns a triples data structure for all dependencies with the word and Part-of-Speech tag for the connected words as well as an identifier of the type of dependency. These are then mapped to objects and actions. Actions are extracted firstly based on them being POS-tagged as verbs, after which it gets its attributes, such as which object performs this action, based on what condition etc. Objects are extracted in two ways. First, they are found by them performing an action, as defined by having a Nominal Subject dependency to a verb. The second extraction of objects is done using the Determiner dependency, describing a relation between a noun and its determiner. The object of an action and the object that performs the action is established by analysing the Nominal Subject dependency between the verb that represents the action and a dependent Noun or a Proper Noun in the case of that object being, for example, a name.

To model additional information on how an action should be performed, parameters are implemented. The parameters considered here are “how”, “with”, “target” and “direction”. A parameter is labelled as “how”, defining how an action is performed, if it is in an Open Clausal Complement dependency and are POS-tagged as an adjective. Both the “with” and “direction” parameters are found in Nominal Modifier dependencies, with the To POS-tag indicating direction and the Preposition or Subordinating Conjunction POS-tag indicating a “with” parameter. The “target” parameter is found in Direct Object dependencies where the target is a noun.

Direction is also found in Phrasal Verb Particle dependencies, together with loop identifiers such as “twice”. In this case, a list of loop identifiers; “once”, “twice” and “thrice” are used to match against to find loop identifiers while a list of directions such as “upward”, “west”, “forward” etc. are used to find directions. The directions are then normalized to “up”, “right”, “down” and “left”.

In this implementation, repeat-actions, or Loops, are handled on an individual action level. This is due to the difficulty of understanding sentences such as “Walk forward

and dance seven times". Should both these actions be performed seven times or just the last one? In addition to finding loop identifiers as Multiplicative Adverbs, Direct Object dependencies with Plural Nouns that also have a Numeric Modifier dependency to a Cardinal Number POS-tag are used. This is based on instructions such as "Walk forward seven times" where "times" would be a plural noun that also has the numerical modifier "seven". This would also pick up sentences like "Greg should eat seven potatoes." and "Greg should eat a potato seven bananas." as "eat" being the action that should be performed seven times. To take into consideration numbers such as "two hundred", a check is also made to see if the cardinal number have a compound dependency to another cardinal number, in that case, these two are concatenated.

Actions also hold conditional statements. These can be in the form of While statements, Until statements, Unless statements and If statements and are found using the Adverbial Clause Modifier dependency while also matching these with Marker and Nominal Subject dependencies.

3.7 Study 2: Semantic Role Labelling

Semantic role labelling is based around verbs and the semantic roles connected to that verb. With the assumption discussed earlier in the method section, that verbs can be interpreted as actions, this approach builds its actions systematically from these verb-structures. The implementation used in this study labels these roles according to the PropBank standard, with numbered and unnumbered arguments representing their impact or relation to a specific verb. The specific meaning of these numbered arguments depends on the specific verb, or rather a specific semantic meaning of a verb, but some generalisations can be made. Looking at many of the verbs that would represent likely actions in this study, Argument 0, Arg0, often represent the Agent, the executor of that action. This is, in this approach, used to identify the object that performs the action. Argument 1, Arg1, is often the Patient, or the target of the action in our case. The rest of the numbered arguments often have more verb-specific semantic meanings, but initial testing has shown that Arg2 often represent the Instrument and A4 often indicates a goal. By then looking for noun POS-tags in this argument this model can find the actions "with" and "target" parameters respectively.

The unnumbered argument ArgM-MNR describes how the verb is executed, and is mapped to the "how" argument of the action. The ArgM-DIR that describes some directional property of a verb is passed through a function that tries to map it to one of four directions: up, right, down and left. If it fails, it is assumed that the directional property is a sentence of type "to an object". In this sentence, it then looks for a noun and if one is found, it is assigned as the "target" property of the action. To find loops, more specifically for-loops and until-loops, the ArgM-TMP role is used.

The for-loops are found through Cardinal Numbers as well as Multiplicative Adverbs by running them through a mapping function that tries to map them to integers, if it succeeds that integer is assigned as the loop iteration counter. The until-loops are found

by looking at the POS-tags trying to find a prepositional tag in conjunction with a noun and an adjective or verb of the type non-3rd person singular. These are then structured into a conditional statement for the action. Other conditional statements, such as if-statements are found using the ArgM-ADV role, that initial tests showed, in the context of this study, accurately represented conditional statements of actions.

For example, in the sentence "the character should walk forward twice", "walk" is the central verb. To this verb, "the character" is labelled as the semantic role Arg0, or the Agent, "forward" as ArgM-DIR and "twice" as ArgM-TMP. This results in the action "walk" containing "character" as the "On object" parameter, two as a loop indicator and "right" as the "direction" parameter.

A difference in this approach to approach one is that Semantic Role Labelling can label chunks of several words as an argument, while Dependency Parsing uses a single-word-resolution. To address this, Part-of-Speech tagging is used to extrapolate the relevant words in these chunks as well as for identifying specific action parameters, for example for separating ArgM-DIR into the "direction" or "target" parameter.

3.8 Semantic matching

According to the findings of Sammet (1966) and Good (2016), one of the larger problems of programming with natural language is the conflict of the ambiguity of the natural language and the strict nature of programming languages. To address this, following the first design principle of Good (2016), semantic matching is used to map the natural language to a limited domain of expressions, in this study referred to as the environment. An overview of the system can be seen in *Figure 4*.

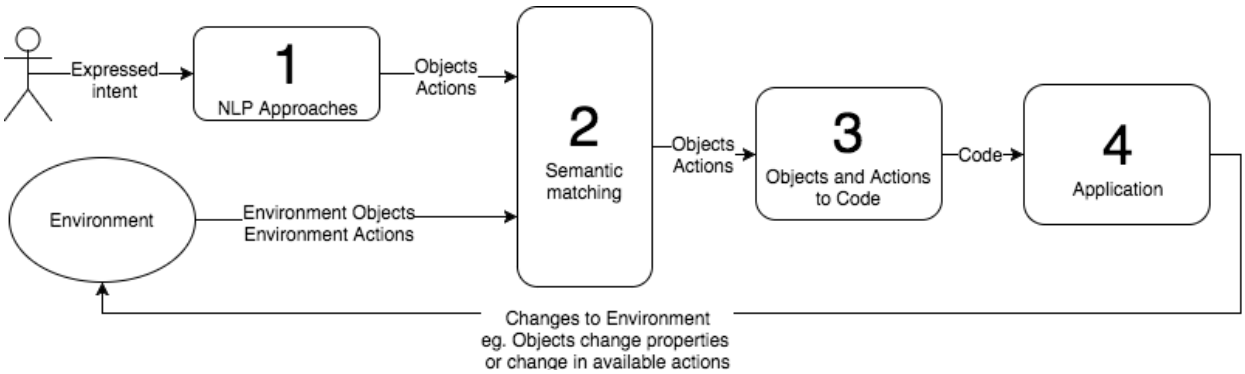

Figure 4: Implementation flowchart

After the different NLP approaches have generated their Objects and Action, these are then semantically matched. The semantic matching is done using the lexical resource WordNet and path similarity as implemented in NLTK. WordNet has, as described earlier, sets of cognitive synonyms, called Synsets. By finding the Synsets for the Environment Actions and Environment Objects and for each action and object parsed

from the natural language given by the user, it is possible to find what Environment Action or object is most similar to. This makes it possible for the system to understand what object or action in the environment the user is referring to and thereby set the correct name and parameters for each object and action.

A problem that arose during initial testing is that, although WordNet has a vast amount of Synsets, it does not cover all words. Some verbs, but primarily many proper nouns, do not have Synsets, making it difficult to use this approach to match them to environmental actions and objects. Only when the proper noun is explicitly defined as the name of an environmental object will WordNet find a match. Defining an environmental object with a proper noun not found in WordNet makes it unable to semantically matched with other words, rendering this function redundant in those cases.

During this stage, to account for the incompleteness of natural language instructions discussed by Good (2016) and Miller (1978), additional, non-semantic matching is made. This matching consisted of finding potentially missing mandatory attributes of actions and updating attributes of object based on actions and objects specified in the environment.

3.9 Code generation

After the semantic matching the code is generated from the resulting *final objects* and *final actions*. In this implementation, the code generated follows JavaScript standard to be able to run in a web browser, but because of its modularity this function could be replaced to generate code in any programming language with an object-method based structure.

As the goal of this implementation is to relay intentions, the main focus of the code is the actions that should be performed. The code generation in this implementation goes through the list of final actions and one by one generates a block of code based on that action. First it identifies if the action has any conditions, and if so, they are formatted to match the code language standards. Secondly the loop-parameter is formatted. As described earlier, due to the problem of accurately parsing if several actions should be performed in the same loop, the generated loops only contain a single action, as does the conditional statement.

As the surrounding code is generated, it identifies the object that is performing the action and, together with the actions parameters, formats the code for calling that action. The action parameters for a specific action are defined in the environment action along with the default values that are overridden if they are also defined by the user. To maintain consistency, these parameters are listed in a specific order where each parameter is optional if not defined in the environment. This order is; “target”, “with”, “direction” and “how”.

4. Results and Discussion

4.1 Ease of implementation

The first part of the implementation for both approaches is the analysis of how human users would describe their intent and how the different Natural Language Processing models would interpret them. With this information, models of how the output from the NLP models could be mapped to a structure that would represent the intent of the user.

As described earlier, the implementations of the two different Natural Language Processing approaches used in this thesis are based on different pre-trained models and NLP frameworks as well as wrapped in different interfaces, making it difficult to compare them side by side. With that said, several generalisations regarding the data returned from the different models and how this data can be structured and combined can be made independently from framework or interface.

The first approach, implementing the Stanford Dependency Parser, returns a data structure containing the dependencies as a dependency label and the two words connected by that dependency and their respective Part-of-Speech tag. Due to the hypothesis of this approach, that the individual dependencies or combinations of dependencies represent the intent of the user, without a higher-level structure, these dependencies become the basis for modelling the user intent to a data structure. Finding objects, actions and how they relate to each other is then a matter of finding patterns in the POS-tags and dependencies that relate to different programming concepts. This gives a greater control, making it possible to find pattern dependent on combinations of several dependencies and tags. It is however, when these patterns become complex, sometimes problematic to find what verb, or action, that the dependencies are originating from. These chains of dependencies also make this approach prone to errors. If one dependency in these complex patterns is missing or mislabelled, this breaks the whole chain.

The models used in the second approach, the Semantic Role Labelling, return the verbs and their semantic roles as a data structure based on the central verb. This, in contrast to the first approach, takes care of relating the patterns to the affected verb. Since this thesis is based on the hypothesis that verbs represent actions, that in turn are dependent on the context of that verb, such as what entity performs that verb, this makes for a well-suited structure of objects and actions. One thing that the Semantic Role Labelling model does not take into consideration is the Part-of-Speech tags. These are added using a separate POS-tagger and can be used to both identify objects mentioned in the text but also give more clarity to the semantic roles, that in many cases can consist of several words. With the combined Semantic Role Labelling and POS-labelling combined, it is possible to find patterns that both rely on the semantic roles related to the verbs but also the relationship between different POS-tags within these roles. Due to the similarity between the output of the NLP models used in this approach and the action-object

structure implemented in this study, this approach seems better suited for implementing intent to code translation.

4.2 Amount of generalisation

The framework developed by Brennan and Resnick (2012) to describe the different programming concepts is the basis for the different approaches developed in this thesis. These concepts are: Sequences, Loops, Events, Parallelism, Conditionals, Operators and Data. Some of these concepts are not implemented due to time constraints and are left for future research, such as Events. Other concepts that are in themselves not directly related to giving instructions, such as the Data, Operators and Parallelism concept have also not been implemented.

One concept that is successfully implemented is Sequences, that one instruction should be executed after another. In the case of Semantic Role Labelling, this is easy to implement due to the verb structures generated from the NLP model that structures the actions together with their roles, regardless of if there are one or several instructions given by the user. In the case of Dependency Parsing, this is done by first finding the verbs from POS-tags and then observing the chains of dependencies that connect to that specific verb. The issue mentioned in previous section, to track dependencies back to a specific word, become increasingly difficult with an increasing amount of instructions given per sentence. This results in that Semantic Role Labelling is a more flexible solution for implementing this programming concept.

The concept of Conditionals is partially successfully implemented; if-statements are implemented for both NLP approaches with similar results. Else and "else-if" statements are not. These concepts cannot consistently be expressed as a pattern of either grammatical dependencies or semantic roles with current NLP models. The if-statements in themselves are structured in the form of an object that either does or does not have a certain property to either perform or not a certain action at that time. Conditionals are also used as a part of Loop-statements.

Loops, being "repeating an action during some condition" are, according to the initial testing, successfully implemented in the case of for-loops, while-loops and until-statements. For-loops represent an action that should be executed a given number of times, often expressed as, for example "walk forward twice" or "jump forward seven times". While-statements, and its counterpart until-statements, will perform an action while, or until, a condition is met. These conditions follow the same structures as the previously mentioned Conditionals.

Apart from the limitations of the implementation of different programming concepts, other limitations of the NLP approaches have been found. One limitation, is inherent to both NLP approaches, is that they can only handle actions that consist of one word, a verb. Even though the semantic role labelling occasionally can find semantic verb frames for word combinations, such as "pick up", these appear inconsistently, are

seldom found in WordNet and are therefore chosen to be limited to the verb, “pick” in this example, for consistency. An alternative approach could be to save the verb as a separate structure in the action to, in the semantic matching, try to find both the combined verb as well as the individual verb action in the environment, with a priority to the combination. The combinations of NLP approaches studied in this thesis also have a limited understanding of the context within the sentence. For example, occurrences of “it” and “itself”, as references to an object mentioned earlier in the sentence, cannot, using these NLP models, be correctly correlated to these objects.

Table 5: Result of qualitative implementation analysis

	Dependency Parsing	**Semantic Role Labelling**
Ease of implementation	Intermediate	Good
Amount of generalisation	Intermediate	Intermediate

4.3 Learnability

During the user testing, learnability was assessed via the amount of additional instructions that were needed during the test. All users were initially given the same information about how to use the system, what objects and actions were available and the task they were supposed to complete. Generally, from observing the participants, they could all quickly get started using the system without any additional information. Most users however had to be reminded, after they had written an instruction or two, to use full, grammatically correct, sentences. Otherwise they tended to shorten their input to the minimal amount of words. For example, one user ended up writing instructions such as “walk tree” when their intention was for the character to walk to the tree. The same user, who had intermediate programming experience, also noted that experience writing instructions in the form of code heavily influenced the way this user gave instructions in natural language. The user said that, by reading the code generated from writing instructions in natural language, the user tended to adapt their instructions to the structure of the code, rather than natural language grammar. The users with novice programming experience showed similar tendencies, to use minimal instructions. The expert users in this study, on the other hand, surprisingly tended to use correct grammar, not adapting to “programming grammar”. The system could interpret most of the shortened sentences but, as was shown by the difference in misinterpreted instructions, giving grammatically correct sentences resulted in higher accuracy. Additionally, one user had to be informed not to use digits, and instead write the name of the number when describing the action they wanted to be performed several times, since digits are not supported in the current implementation.

In the survey following using the system, most users responded that they thought that the instructions on how to use the system were easy to understand.

Table 6: Result of qualitative analysis of Learnability

	Dependency Parsing	**Semantic Role Labelling**
Novice users	Intermediate	Intermediate
Intermediate users	Intermediate	Intermediate
Expert users	Good	Good

4.4 Efficiency

By measuring the time, it took for the users to complete the task of moving the character object to the goal in the game environment and then observing while the users later experimented with alternative solutions, it was possible to get a measurement of the efficiency of using the system. All users were increasingly efficient as they got more familiar with the system and all users managed to complete the task faster after experimenting with different approaches for a while. When it came to the initial performance, there was little difference between the novice and intermediate users, where both groups had longer times to completion, ranging from three minutes up to four and a half minutes. The expert users on the other hand showed good initial performance with time to completion close to one minute. This could potentially be because of their tendency to, to a higher degree, use grammatically correct sentences but also their ability to understand the generated code and find patterns in the mapping between expressed commands and generated code. Between the two NLP approaches; Semantic Role Labelling and Dependency Parsing, there were no detectable difference in efficiency. Because of the limited number of test subjects and because of the time constraints of this study, it is not possible to say how strong the correlations between the programming experience, NLP approaches and efficiency are. The results found here could possibly be seen an indication of that the different NLP approaches are equivalent when it comes to efficiency and that higher level of programming experience does, to some degree, have an influence on the efficiency of using the system.

Table 7: Result of qualitative analysis of Efficiency

	Dependency Parsing	**Semantic Role Labelling**
Novice users	Bad	Bad
Intermediate users	Bad	Bad
Expert users	Good	Good

4.5 Errors

During the time the users solved the task of moving the character past the obstacles and to the goal, the number of commands that the system either did not understand or misinterpreted were recorded. From this data, a strong inverse correlation could be found between the amount of programming experience and the number of total errors. The novice users had a high amount of total errors but the type of errors they had differed, depending on the NLP approach. The novice user using Semantic Role Labelling had only a few instructions that could not be interpreted but a high amount of instructions that were misinterpreted. The other user in the novice category, using Dependency Parsing, had a high amount of instructions that could not be interpreted but fewer that were misinterpreted. From observing these users, the total amount of errors can be seen as partially coming from their inability to understand the code that is generated and its correctness in regard to their given instruction. The system has been shown to not have hundred percent accuracy and flexibility. This makes the user's the ability to understand the correlation between natural language input and code output, have a great impact on the amount of errors the users encountered in this study.

Between the two different NLP approaches Dependency Parsing resulted in more errors overall, across all levels of programming experience with a higher rate of instructions that could not be interpreted by the system.

When reviewing the log-files after each user test, it was possible to understand the cause of many of the system errors that occurred during the test. The most common cause of instructions that could not be interpreted was that the described action was not labelled as a verb by the Part-of-Speech tagger, breaking the assumption that an instruction consists of an action with related objects. This happened frequently when the user gave incomplete or grammatically incorrect sentences but was also observed when the user input was seemingly correct. In the case of Dependency Parsing this happened often when one user gave the instructions in present tense. For example, when the user gave the instruction "The character walks to the tree", the system labelled "walks" as a plural noun rather than a verb, making the system unable to process the instruction. In the case of another user, also using Dependency Parsing, "jump" and "move", was also

mislabelled as a noun rather than a verb, even though used in grammatically correct sentences. Looking at the results from user tests with users using Semantic Role Labelling, they had lower occurrences of these types of mislabelling. The system showed a larger flexibility when using Semantic Role Labelling by accurately handling instructions in present tense as well as descriptions of what should happen, such as “the character should walk to the tree” or “walk to the tree”.

One thing that both approaches struggled with was correctly interpreting directions of actions. The direction “left”, as in the instruction “the character should walk left”, was mislabelled with high frequency as a verb, while many of the other directions, such as “downwards”, were often mislabelled as nouns. In the Semantic Role Labeller used in this thesis there exists a specialised label for direction, ArgM-DIR. During this study, this label was, unfortunately, not consistently assigned to the directions mentioned in the instructions, possibly because of the discussed mislabelling of POS-tags. Semantic Role Labelling also tended to, but not consistently, as discussed earlier, use combined verbs. These combined verbs have been shown to sometimes be a verb-direction combination, such in the case of “jump down” or “walk down”. As described earlier in this thesis, to maintain consistency without overfitting, these combined verbs were reduced to their single verb word. In the case of directions, this has occasionally been shown to, unfortunately, discard valuable information.

Another issue that was discovered during the user tests was during the implementation of loops. As described in the *Amount of generalisation* chapter, the results from the initial testing during the development phase showed great promise for implementation of the loop concept. During the user testing however, it was apparent that the implementation of the loop concept did not cover all cases. Not all users took advantage of this concept and those who did, more often than not, had errors associated with it. These users primarily used the loop concept, together with the actions “walk” and “jump”, as a measurement of the distance rather than an indication of how many times the action should be performed. For example, “walk two steps forward” or “jump four squares to the right” rather than “walk forward two times” and “jump right two times” (as the jump action moves the object two squares). Due to the nature of the specified domain, the accurate representation of these intents would most likely be in the form of loops. But since the part of the system that turns the intent into actions and objects, by design, is decoupled from the application, there is no correlation between actions and measurement of distance. This also does not follow the assumptions of loop usage considered in this thesis and its possibility to be interpreted by the different NLP models has therefore not been studied.

Sequences, the ability to give instructions containing several actions that should be performed in a specific order, also showed great promise during the development phase. This concept was used by most participants, ranging across all levels of programming experience, with more experienced users tending to express a larger amount of instructions in the same sentence. The system could handle most instructions with two

actions per sentence but with three or more actions per sentence, the accuracy decreased rapidly. Dependency Parsing had some issues relating parameters, such as who performed an action or how that action should be performed, to the correct action, even in sentences containing only one or two actions. Semantic Role Labelling showed better results in this regard, with the verb frames creating separate data structures for each action. It did however suffer from a decline in accuracy with three or more actions per instruction. One user, who had an expert level of programming experience, tried using five instructions in one sentence; "Character walk to tree and cow walk to tree and cow eat tree and cow walk up and character walk to goal". In the case of this sentence, the system showed a high amount of mislabelling such as two of the instances of "walk" being labelled as nouns. The lowering accuracy of the NLP models with increasing amount of actions per sentence could potentially be due to the data they are trained on. As sentences in the data sets used for these models, Penn Treebank, English Wikipedia and RCV1, as in the English language in general, include only a few verb phrases per sentence, it can be assumed to influence the accuracy. It is therefore possible to theorize that, given enough data, higher accuracy for the parser could be achieved even with a higher amount of actions per instruction.

It was also observed that the user referred to each object explicitly in each action in the five instructions in one sentence example given above. Other users tended to often refer to "it" or "itself" as in the sentences "Character should move to the tree and eat it" and "the cow should eat itself". Using the implementation used in this thesis, "it" or "itself" could be identified as being the target of the action but it was not possible to find a pattern in the data from the NLP models that would indicate what they referred to.

One key feature for errors in usability is the possibility to be able to easily correct them. In this study it was clear that, to be able to correct errors in the implementation, an understanding of programming is required. From the three different levels of programming experience in this study; novice, intermediate and expert, the novice users had a lot of problems understanding and correcting the errors that occurred. These users showed no, or very limited, understanding of the code that was generated from their natural language instructions. They were only able to judge the correctness of the system by executing the code in the game environment and observing what happened. Since the system is trying to correct semantic differences by running the objects and actions through Semantic Matching, if the output of the NLP model is incorrect, these incorrect objects and actions will then be semantically matched, often creating more confusion. The users with more programming experience could detect errors in the generated code before executing it and therefore use alternative wording, or correct grammar and/or spelling in their given instruction. These users did, however, have the same problem as the novice users when it came to incorrect NLP output being semantically matched, creating confusing output.

Table 8: Result of qualitative analysis of Errors

	Dependency Parsing	**Semantic Role Labelling**
Novice users	Poor	Poor
Intermediate users	Poor	Intermediate
Expert users	Intermediate	Good

4.6 Satisfaction

After the users completed the predefined task and were given some time to experiment with the system, they responded to a series of questions regarding their thoughts on using the system. These questions were on a disagree-agree scale ranging from one to five. When asked if they found the system fun to use, most users responded positively, with a mean value of four. This could potentially be due to the game aspect of the interface, as described by Bromwich, Masoodian & Rogers (2012), making it engaging and fun to use.

Another aspect that the users agreed on was that the system could not accurately interpret their intentions, with a mean score of two out of five. This confirms the low accuracy found in the data from the user tests of the system. As one user commented; "I think [this approach to programming] could work, but the system is very fragile, misunderstandings can feel very frustrating". The user also mentioned the concept of Uncanny valley in robotics, the theory that robots must look and behave very close to, or very different from real humans or humans will have strong negative emotions towards them. "Almost human" is not good enough. Much in the same way, programming by intent, as in this thesis, according to this user, will need to have an almost hundred percent accuracy and a great deal of flexibility to make it pleasant and natural to use. This can possibly have some impact on the perceived influence on the Computational Thinking of the users. When asked about if the system contributed to their understanding of how to instruct a computer to solve problems, the general opinion was divided with a slight majority towards the no side, resulting in a mean score of two point five. When instead asked if the system had contributed to their understanding of programming, the mean value was slightly higher with a score of three. One user commented that "When programming I am used to write short commands and often try to keep the code as short as possible. Therefore, it was a bit difficult getting used to writing longer commands". This might indicate that, with already established knowledge of programming, writing code might be a more efficient way of communicating intent to the computer. This might also have influenced that users overall did not feel that the system was easy to use, with a mean value score of two point sixty-seven.

Table 9: Result of qualitative analysis of Satisfaction

	Dependency Parsing	**Semantic Role Labelling**
Novice users	Intermediate	Intermediate
Intermediate users	Intermediate	Poor
Expert users	Poor	Intermediate

5. Conclusions and future work

In this thesis, the possibility of using Natural Language Processing techniques to translate human intent to code, making programming more accessible, has been studied. Two different approaches have been implemented, based on combinations of the NLP techniques: Part-of-Speech tagging, Semantic Role Labelling and Dependency Parsing. Based on these two approaches, two hypotheses were formed regarding how these techniques could be used to extract the intent of instructions given in natural language.

Hypothesis 1*: It is possible to, using Part-of-Speech tags and grammatical dependencies, model a sentence of human intent as a set of objects, methods and their relations. By structuring these object and methods by their relations, code can be created, representing the user's intent.*

From the study done in this thesis, the findings indicate that this hypothesis is the weakest of the two proposed. The user tests using this approach had a higher amount of instructions that could not be interpreted. This was partly due to the overall slightly lower labelling accuracy observed by the models implemented in this approach and a lower observed flexibility when it comes to tenses. Regarding the generalisation of using grammatical dependencies and Part-of-Speech tags to model intent, this was tightly coupled with the models themselves, making it difficult to assess its feasibility without further research. Given the low accuracy of the models, the assumptions made in this hypothesis can neither be confirmed or rejected.

Hypothesis 2*: It is possible to, with Semantic role labels and Part-of-Speech tags, model a sentence of human intent as a set of objects, methods and their relations. By structuring these object and methods by their relations, code can be created, representing the user's intent.*

The NLP models applied in this hypothesis had slightly higher accuracy and flexibility. Using Semantic Role Labelling, the NLP models also returned data in a structure that was highly compatible with the action-object data structure implemented in this thesis, making implementation of these models more efficient. This approach also showed more promising results with multiple actions mentioned in one sentence, even though, looking at the results in this study, additional data and training of these models would be needed. As with the first hypothesis, the close relation between the labelling accuracy of the models and the general approach of using semantic role labels to model intent, makes it difficult to determine its feasibility. Additional research is therefore needed to confirm or reject this hypothesis.

The accuracy of the models used in both approaches have shown to be of great importance. Low accuracy caused frustration and confusion among the users in this study. The observed low accuracy has a large impact on the possibility to apply this way of programming to improve Computational Thinking. According to the findings in this study, many users experienced problems with the system not interpreting their intent correctly and that the system did not, to any larger extent, contribute to their understanding of Computational Thinking or programming. As described one user; for this way of turning intent to code to be successful, close to a hundred percent accuracy is most likely needed to be reached. Given that the system could reach a hundred percent accuracy and a great deal of flexibility, it could then be possible for novice users to use the system. In its current state, initial knowledge about programming is more or less required to use the system. The low accuracy also made the user experience of the system troublesome even for experienced programmers.

One assumption made in this thesis, that programming with intent expressed in natural language would benefit from Semantic Matching, was in many occurrences confirmed but in the cases where labelling errors occurred in the NLP models, this caused an extra level of confusion. A solution to this might be to query the user regarding the assumptions that the system makes instead of feeding the results to the user immediately. However, in accordance with the design principles of Good (2016), it is important so avoid giving a sense of the system being more capable than it is when interacting with the user.

This thesis, due to the time limitations, was not able to conduct large scale studies on the effect on Computational Thinking. The user testing in this study was conducted with adult users. Additional studies would have to be made with larger test groups and a variety of age groups to more accurately determine the effect this type of programming have on the possibility to learn Computational Thinking at different ages. As this thesis has shown, the low accuracy of the NLP models would first be addressed, as it strongly affects the usage of the system and especially makes it troublesome for those without prior programming knowledge.

The work in this thesis lays a broad foundation and aims at a great impact on further research on how Natural Language Processing can help make technology more accessible to everyone. The current findings about the performance of different NLP methods in this study are hoped to be a solid starting point for research and development into intent extraction from natural language.

As for other future work on the subject, but beyond the scope of this project, there are several things that could be focused on. In this thesis, the English language was used as a basis for finding correlations in the form of semantic roles and dependencies. Even though it is hypothesized in this thesis that these techniques, given similar tools, models and lexical resources, could be applied to other languages, this is something that needs to be studied. Additionally, some of the errors that occur in the system developed in this thesis were due to inaccuracy in the Natural Language Processing models. By training models on more data, specific to this use case, the overall accuracy of the system could potentially be increased.

A shortcoming in both the NLP approaches studied in this thesis is their inability to fully account for context, for example mentions of "it" and "itself". This could potentially be addressed using additional Natural Language Processing techniques, making it also subject for further research.

Future work could also be done on the form of alternative implementations of these techniques. For example, by studying alternative representations of intent other than objects and actions, but also by studying how the techniques could be applied in other domains, other than a game interface. Additionally, implementing a voice interface for expressing intent to the computer could be studied to assess if this would have an impact on how the users expressed their intent, possibly being more verbose than in the text interface implemented in this thesis. Further, this thesis did not include all the programming concepts discussed by Brennan and Resnick (2012); Sequences, Loops, Events, Parallelism, Conditionals, Operators and Data. By having an alternative domain, the implementation of all the concepts or another subgroup of the concepts could be studied.

Additional research and development in the area of Natural Language Processing is needed to create models with higher accuracy to determine the potential for translating human intent to code. If higher accuracy could be achieved, systems like the one developed in this thesis could further be the subject of further research to determine their impact on Computational Thinking and programming. If higher accuracy cannot be achieved, the system will most likely create confusion and frustration, being especially difficult to use for novice users as observed in this study. Although the research presented in this thesis can be only considered exploratory, it does give some exciting insights into how natural language may eventually be translated into fully functional code. Such an advance should be able to address the anticipated future shortfall in the number of people with programming skills.

References

Abbott, R. J. (1983), “Program design by informal English descriptions”, Communications of the ACM CACM Homepage Archive Volume 26 Issue 11, ACM: New York, NY, USA.

Abebe, S.L. & Tonella, P. (2010), "Natural Language Parsing of Program Element Names for Concept Extraction", page 156.

Adams Becker, S., Cummins, M., Freeman, A., and Rose, K. (2017), 2017 NMC Technology Outlook for Nordic Schools: A Horizon Project Regional Report. Austin, Texas: The New Media Consortium.

Adetunmbi, O.A., Obe, O.O. & Iyanda, J.N. (2016), Int J Speech Technol (2016) 19: 929.

AlSuhaibani, R.S., Newman, C.D., Collard, M.L. & Maletic, J.I. (2015), "Heuristic-based Part-of-Speech tagging of source code identifiers and comments", IEEE, page 1.

Ayetiran, E. F., Boella, G., Di Caro, L. & Robaldo, L. (2015), *“Enhancing Word Sense Disambiguation Using a Hybrid Knowledge-Based Technique”*, Natural Language Processing and Cognitive Science, edited by Olga Acosta, et al., De Gruyter, 2015. ProQuest EBook Central.

Biermann, A.W., Ballard, B.W. & Sigmon, A.H. (1983), "An experimental study of Natural Language Programming", International Journal of Man-Machine Studies, vol. 18, no. 1, pp. 71-87.

Bohnet, B. (2011), “Comparing Advanced Graph-based and Transition-based Dependency Parsers”, International Conference on Dependency Linguistics, Depling 2011, Barcelona, September 5-7, 2011.

Brennan, K. & Resnick, M. (2012), “Using artifact-based interviews to study the development of Computational Thinking in interactive media design”, American Educational Research Association Meeting. Vancouver, BC: Canada.

Bromwich, K., Masoodian, M. & Rogers, B. (2012), “Crossing the Game Threshold: A System for Teaching Basic Programming Constructs”, CHINZ '12 Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction, Pages 56-63.

Brill, E. (1995), “Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging”, Computational Linguistics: Volume 21 Issue 4, December 1995, page 543.

Capindale, R. & Crawford, R. (1990), “Using a natural language interface with casual users”, International Journal of Man-Machine Studies, Volume 32, Issue 3, Pages 341-361.

Carreras, X. (2007), “Experiments with a Higher-Order Projective Dependency Parser“, Conference: EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic.

Chen, D.L. & Mooney, R.J. (2011), "Learning to Interpret Natural Language Navigation Instructions from Observations", Proceedings of the 25th AAAI Conference on Artificial Intelligence, San Francisco, CA, USA.

Chen, D. & Manning, C. (2014), “A Fast and Accurate Dependency Parser Using Neural Networks.”, Proceedings of EMNLP 2014.

Code.org, (2014), “Code.org Overview”, available online: https://code.org/files/Code.orgOverview.pdf (2017-04-20).

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. & Kuksa, P. (2011), “Natural Language Processing (Almost) from Scratch”, Journal of Machine Learning Research (JMLR).

Falleri, J., Huchard, M., Lafourcade, M., Nebut, C., Prince, V. & Dao, M. (2010), "Automatic Extraction of a WordNet-Like Identifier Network from Software", page 4.

Foth, K., Daum M., & Menzel W. (2005), “Parsing Unrestricted German Text with Defeasible Constraints.” In: Christiansen H., Skadhauge P.R., Villadsen J. (eds) Constraint Solving and Language Processing. CSLP 2004. Lecture Notes in Computer Science, vol 3438. Springer, Berlin, Heidelberg.

FrameNet, (2017), “What is FrameNet?”, Berkeley University of California, available online: https://framenet.icsi.berkeley.edu/fndrupal/WhatIsFrameNet (2017-02-14).

Gales, M.J.F., Kim, D.Y., Woodland, P.C., Chan, H.Y., Mrva, D., Sinha, R. & Tranter, S.E., (2006), "Progress in the CU-HTK broadcast news transcription system," in IEEE Transactions on Audio, Speech, and Language Processing, vol. 14, no. 5, pp. 1513-1525, Sept. 2006.

Galliano, S., Geoffrois, E., Mostefa, D., Choukri, K., Bonastre, J-F., & Gravier, G. (2005), “The ESTER phase II evaluation campaign for the rich transcription of French broadcast news”, Proceedings of the 9th European Conference on Speech Communication and Technology (INTERSPEECH’05), pp. 1149– 1152, 2005.

Good, J., (2011), “Learners at the wheel: novice programming environments come of age.”, International Journal of People-Oriented Programming, 1(1), 1e24.

Good, J., (2016), “Programming language, natural language? Supporting the diverse computational activities of novice programmers”, Journal of Visual Languages and Computing.

Google (2017), “Cloud Speech API - Speech to text conversion powered by machine learning”, Google website, available online: https://cloud.google.com/speech/ (2017-04-03).

Grudin, J. (1992), “Utility and usability: research issues and development contexts”, Interacting with Computers, Volume 4, Issue 2, August 1992, Pages 209-217.

Hirschberg, J. & Manning C. (2015), “Advances in natural language processing”, Science, American Association for the Advancement of Science, volume 349, number 6245, pages 261-266.

IBM (2017), “Speech to Text - Convert human voice into written word”, IBM Website, available online: https://www.ibm.com/watson/developercloud/speech-to-text.html (2017-03-29).

Kalelioğlu, F. (2015), ”A new way of teaching programming skills to K-12 students: Code.org”, Computers in Human Behavior, Volume 52, November 2015, Pages 200–210.

Kelleher, C. & Pausch, R. (2005), “Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers”, ACM Computing Surveys (CSUR), Volume 37 Issue 2, June 2005, ACM: New York, NY, USA.

Kim, S. & Kim, D. (2016), "Automatic identifier inconsistency detection using code dictionary", Empirical Software Engineering, vol. 21, no. 2, page 565-604.

Kuhn, A., Ducasse, S. & Gîrba, T. (2007), "Semantic clustering: Identifying topics in source code", Information and Software Technology, vol. 49, no. 3, page 230-243.

Levin, B. (1993), “English Verb Classes and Alternations: A Preliminary Investigation”, University of Chicago Press.

Lifelong Kindergarten Group, MIT Media Lab (2017), Scratch. Available online: https://scratch.mit.edu/ (2017-02-15).

Manning, C. & Schütze, H. (1999), “Foundations of statistical natural language processing”, Book, Cambridge, Mass. MIT Press, cop. 1999.

Matinez, A. R. (2012) “Part-of-speech tagging”, Wiley interdisciplinary reviews. Computational statistics, 2012, 4 pages 107–113.

Marcus, M. (1995). New trends in natural language processing: statistical natural language processing. Proceedings of the National Academy of Sciences of the United States of America, 92(22), 10052–10059.

Maruyama, H. (1990), “Structural disambiguation with constraint propagation.” In Proceedings of the 28th annual meeting on Association for Computational Linguistics (ACL '90). Association for Computational Linguistics, Stroudsburg, PA, USA, page 31-38.

McDonald, R., Crammer, K. & Pereira, F. (2005a), “Online Large-Margin Training of Dependency Parsers”, Conference: ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA .

McDonald, R. Pereira, F., Ribarov, K. & Hajič, J. (2005b), “Non-projective dependency parsing using spanning tree algorithms”. In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT '05). Association for Computational Linguistics, Stroudsburg, PA, USA, 523-530.

Mcdonald, R. & Pereira, F. (2006), “Online Learning of Approximate Dependency Parsing Algorithms”, Conference: EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy.

Menzel, W. & Schröder, I. (1998), “Decision procedures for dependency parsing using graded constraints.” Proceedings of the workshop on processing of dependency-based grammars (ACL-COLING), page 78–87.

Microsoft (2016), “Bing Speech API - Convert audio to text, understand intent, and convert text back to speech for natural responsiveness.”, Microsoft website, available online: https://www.microsoft.com/cognitive-services/en-us/speech-api (2017-03-29).

Miller, L.A. (1978), “Behavioral studies of the programming process”, Tech. Rep., IBM Thomas J Watson Research Center, Yorktown Heights, NY.

Navigli, R. (2009), “Word Sense Disambiguation: A Survey”, ACM Comput. Surv. 41, 2, Article 10, ACM: New York, NY, USA.

Nielsen, J. (1993), “Usability Engineering”, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Nivre, J. & McDonald, R. (2008) “Integrating Graph-Based and Transition-Based Dependency Parsers”, Proceedings of ACL-08: HLT, pages 950–958, Columbus, Ohio, USA, June 2008.

Nivre, J (2010), “Dependency Parsing”, Language and Linguistics Compass 4/3 (2010): 138–152.

Nouvel, D., Ehrmann, M. & Rosset, S. (2016), “Named Entities for Computational Linguistics”, John Wiley & Sons, Incorporated, 2016. ProQuest EBook Central.

Palmer, M., Gildea, D. & Xue, N. (2011), 'Semantic Role Labelling', 1st edn, San Rafael, Calif., 1537 Fourth Street, San Rafael, CA 94901 USA, Morgan & Claypool Publishers, 2011.

Pollock, L., Vijay-Shanker, K., Shepherd, D., Hill, E., Fry, Z. & Maloor, K. (2007), "Introducing natural language program analysis", ACM, page 15.

PractNLPTools (2016), PractNLPTools, Available online: https://github.com/biplab-iitb/practNLPTools (2017-02-22).

Princeton University (2010), "About WordNet." WordNet. Princeton University. Available online: http://wordnet.princeton.edu (2017-02-15).

PropBank, (2017), “The Proposition Bank (PropBank)”, available online: http://propbank.github.io/ (2017-02-14).

Ratnaparkhi, A., (1996), “A maximum entropy model for Part-of-Speech tagging”. EMNLP 1, pages 133–142.

Regeringskansliet (2017), “Stärkt digital kompetens i läroplaner och kursplaner”, available online: http://www.regeringen.se/pressmeddelanden/2017/03/starkt-digital-kompetens-i-laroplaner-och-kursplaner/ (2017-04-20).

Saeed, J., (2015), “Semantics”, Wiley, ProQuest EBook Central.

Sáez-López, J.-M., Román-González, M. & Vázquez-Cano, E., (2016), “Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools”, Computers & Education, Volume 97, June 2016, Pages 129–141.

Sammet, J.E., (1966), “The use of English as a programming language”, Communications of the ACM CACM Homepage archive, Volume 9 Issue 3, ACM: New York, NY, USA http://dl.acm.org.ezproxy.its.uu.se/citation.cfm?id=365274.

Sampson, G. (1980), “Schools of Linguistics”, London: Hutchinson & Co.

Scheinberg, S. (1960) “Note on the Boolean properties of context free languages”, Information and Control, Volume 3, Issue 4, 1960, Pages 372-375.

Sekine, S. & Ranchhod, E., (2009) “Named Entities”, John Benjamins Publishing Company, 2009. ProQuest EBook Central.

SemLink, (2013), “SemLink”, University of Colorado, Available online: https://verbs.colorado.edu/semlink/ (2017-02-15).

Shepherd, D., Pollock, L. & Vijay-Shanker, K. (2007), "Case study: supplementing program analysis with natural language analysis to improve a reverse engineering task", ACM, page. 49.

Shires, G. & Wennborg, H. (2012), “Web speech API Specification”, Speech API Community Group, available online: https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html (2017-03-28).

Stenmark, M. & Malec, J., (2014), “Describing constraint-based assembly tasks in unstructured natural language”, IFAC Proceedings Volumes, Volume 47, Issue 3, 2014, Pages 3056-3061.

Toutanova, K., Klein, D., Manning C. and Singer, Y. (2003), "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network". In Proceedings of HLT-NAACL 2003, page 252-259.

VerbNet, (2017), "A Class-Based Verb Lexicon", University of Colorado, available online: https://verbs.colorado.edu/~mpalmer/projects/verbnet.html (2017-02-14).

Wing, J.M. (2008), "Computational Thinking and thinking about computing", Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 10/2008, Volume 366, Number 1881.

Yle (2015), "Coding soon to be part of Finnish schoolchildren's core curriculum", available online: http://yle.fi/uutiset/osasto/news/coding_soon_to_be_part_of_finnish_schoolchildrens_core_curriculum/7818567 (2017-04-20).