Literature Review Generated by Chat GPT

* [Process](#process)
* [Overview of Contextual Information](#overview-of-contextual-information)
  + [Informal education and non-formal education](#informal-education-and-non-formal-education)
  + [Informal and Non-formal STEM education and family involvement](#informal-and-non-formal-stem-education-and-family-involvement)
  + [Game Making Pedagogies](#game-making-pedagogies)
* [Guiding Concepts and Terms within this Context of this Study](#guiding-concepts-and-terms-within-this-context-of-this-study)
  + [Computational Thinking](#computational-thinking)
  + [Learner Agency](#learner-agency)
  + [Learning Coding from the perspective of Activity Theory](#learning-coding-from-the-perspective-of-activity-theory)
* [Game Making Pedagogies and Tools](#game-making-pedagogies-and-tools)
  + [Code Playgrounds](#code-playgrounds)
  + [Learning Coding through Tinkering](#learning-coding-through-tinkering)
  + [Remixing Games as a Way of Learning Coding](#remixing-games-as-a-way-of-learning-coding)
  + [Constructionist Approaches](#constructionist-approaches)
  + [Universal Design for Learning (UDL)](#universal-design-for-learning-udl)
  + [Computational Thinking Patterns and Scalable Game Design](#computational-thinking-patterns-and-scalable-game-design)
  + [Game Design Patterns](#game-design-patterns)
  + [Game Design Patterns as a way to learn Game Making](#game-design-patterns-as-a-way-to-learn-game-making)
  + [The “Use, Modify, Create” (UMC) approach](#the-use-modify-create-umc-approach)
* [Game Making Communities](#game-making-communities)
  + [Leveraging Funds on Knowledge to create Game Making communities](#leveraging-funds-on-knowledge-to-create-game-making-communities)
  + [The Development of Game Making Communities: Industry-based Resources and Initiatives](#the-development-of-game-making-communities-industry-based-resources-and-initiatives)
  + [Scratch - An On-line Community for Young Coders](#scratch---an-on-line-community-for-young-coders)

## Process

I spent about one hour putting in the following kind of prompts to https://chat.openai.com/chat;

* write 300 words on informal compared to non-formal education . Include references
* tinkering as a pedagogy
* write 400 words on informal science education for families

I then spent about 30 mins trying to find a framework to move from broader ideas to more specific ones.

Finally, I added references to many of the sections in a very piecemeal way. This is an area that Chat GTP struggles with.

## Overview of Contextual Information

### Informal education and non-formal education

Informal education and non-formal education are two approaches to learning that differ in their structure, content, and goals. Informal education refers to learning that takes place outside of a traditional classroom setting, while non-formal education refers to learning that takes place outside of the formal education system, but is structured and intentional (European Centre for the Development of Vocational Training, 2009; Werquin, 2009; Patrick, 2010; Rogers, 2014).

Informal education can take many forms, including experiences, observations, conversations, and life experiences. It is often spontaneous, unstructured, and does not have a specific learning outcome or goal. Informal education is often a natural part of daily life, such as learning about cooking from a family member or learning about a new hobby from friends. Informal education is a valuable way for people to learn and grow, as it can provide opportunities for personal and social development.

Non-formal education, on the other hand, is intentional and structured, but still outside of the formal education system. Non-formal education often takes place in community-based organizations, such as youth groups, libraries, or community centres. Non-formal education can take many forms, such as workshops, training programs, and after-school programs. Non-formal education is designed to meet specific educational needs and is often aimed at specific groups of people, such as youth or adults. Non-formal education can provide opportunities for people to acquire new knowledge, skills, and attitudes in a supportive and structured environment.

Family involvement is a critical component of both informal and non-formal education. Research has shown that parental involvement in a child’s education can positively impact academic outcomes and educational attainment (Henderson & Mapp, 2002). Family involvement can take many forms, such as volunteering at a child’s school, participating in family learning activities, or simply communicating regularly with a child’s teacher.

### Informal and Non-formal STEM education

Informal science education refers to learning about science and technology that takes place outside of formal educational settings, such as schools and universities. Informal science education can be found in a variety of places, such as museums, science centers, zoos, and aquariums, and is designed to be engaging and accessible to a wide range of audiences, including families.

Family involvement in informal science education has been shown to have a number of benefits, including increased knowledge and understanding of science and technology, improved critical thinking and problem-solving skills, and increased interest in science and technology. Family involvement in science education can also help to promote STEM (Science, Technology, Engineering, and Mathematics) literacy, which is becoming increasingly important in today’s rapidly changing world.

Informal science education programs for families often use hands-on, interactive activities and exhibits to engage visitors and encourage learning. For example, museums and science centres may have interactive exhibits that allow visitors to experiment with different scientific concepts, such as electricity and magnetism, or to explore different ecosystems, such as rainforests or oceans. These exhibits are designed to be accessible and engaging, and often use colourful displays and interactive components to make learning fun and enjoyable.

Another important aspect of informal science education is the role that it plays in promoting equity and inclusion in STEM. Informal science education programs can provide opportunities for under-represented groups, such as women and minorities, to learn about science and technology and to engage in hands-on STEM activities. This can help to reduce the gender and ethnicity gaps in STEM and increase opportunities for under-represented groups in these fields.

### Game Making Pedagogies

Game making pedagogies refer to the educational approaches and methods used to teach students how to design and create games. These pedagogies are based on the idea that game making can be an engaging and effective way to learn, as it provides students with opportunities to apply their knowledge and skills in meaningful and creative ways (Kafai, 1994; Kafai and Burke, 2014, 2015, 2017).

One approach to game making pedagogy is to use a project-based learning approach, where students work together to design and create their own game. This allows students to explore their interests, develop their creative and technical skills, and work collaboratively with others. Additionally, project-based learning can help to foster problem-solving skills, critical thinking, and communication skills, as students must work together to overcome challenges and complete their project(Sefton-Green, 2013).

Another approach to game making pedagogy is to incorporate game design principles and elements into the curriculum. For example, students can learn about game mechanics, such as rules, objectives, and challenges, as well as game design elements, such as graphics, sound, and animation. By learning about these elements, students can gain a deeper understanding of the design process and develop the skills they need to create their own games (Kafai and Burke, 2017).

Game making pedagogy can also be used to teach a range of subjects, including mathematics, science, and language arts. For example, students can learn about math concepts, such as probability and statistics, by creating games that incorporate these concepts. Similarly, students can learn about science concepts, such as the laws of motion and energy, by designing games that simulate these concepts (Kafai and Burke, 2014).

Finally, game making pedagogy can be used to develop digital literacy skills, such as coding and computer programming. By designing and creating their own games, students can learn how to write code, create animations, and develop interactive experiences. This can help students to develop the skills they need to succeed in the digital world and prepare them for careers in technology and digital media (Sefton-Green and Erstad, 2012; Sefton-Green, 2013).

## Guiding Concepts and Terms within this Context of this Study

### Computational Thinking

Computational thinking (CT) is a problem-solving approach that involves breaking down complex problems into smaller, more manageable parts and using algorithms, abstraction, and data representation to solve them (Wing, 2008). CT involves a set of skills and processes that can be applied across a wide range of fields and disciplines, including computer science, mathematics, science, engineering, and the humanities(Allan et al., 2010; Brennan and Resnick, 2012; Curzon et al., 2014; Lye and Koh, 2014; Grover and Pea, 2017).

The core elements of CT include:

* Decomposition: Breaking down a complex problem into smaller, more manageable parts.
* Abstraction: Identifying the essential elements of a problem and abstracting away the details that are not relevant.
* Pattern recognition: Identifying patterns and structures in the problem, and using these to develop algorithms and programs to solve it.
* Algorithm design: Developing step-by-step instructions, or algorithms, to solve a problem.
* Data representation: Using data structures and data representation techniques, such as arrays and graphs, to organize and manipulate information.

CT provides a flexible and adaptable framework for solving problems and can be applied to a wide range of real-world scenarios. It is widely recognized as an important 21st-century skill and is increasingly being incorporated into the school curriculum to help students develop critical thinking and problem-solving skills that are relevant to a wide range of fields and industries (Curzon et al., 2014).

### Learner Agency

Learner agency is a term used in education to describe the extent to which learners have control over their own learning and can make decisions about what, when, and how they learn(Toohey and Norton, 2003; Mercer, 2012; Sannino et al., 2016). It is an important concept as it recognizes that learners are not passive recipients of knowledge, but active participants in their own learning process. Learner agency gives individuals the power to direct their own learning journey and make decisions that shape their education and future.

Learner agency encompasses several key aspects of the learning process, including motivation, self-directed learning, and autonomy. A learner with high agency is motivated to learn and takes an active role in the learning process, seeking out resources and opportunities to gain new knowledge and skills. They are also able to direct their own learning and make decisions about what they want to learn and how they want to learn it, rather than being solely guided by the teacher or curriculum(Mercer, 2012).

One way to develop learner agency is to create a learning environment that is supportive and empowering. This includes giving learners access to a variety of resources and opportunities, such as hands-on experiences, project-based learning, and opportunities to explore their interests (Sannino et al., 2016). Teachers can also help foster learner agency by allowing students to take an active role in the learning process, such as setting their own goals, choosing their own learning methods, and reflecting on their progress (Toohey and Norton, 2003).

### Learning Coding from the perspective of Activity Theory

Activity theory is a framework that is used to analyse and understand the nature of human activity and how it contributes to learning and development. The theory views activity as a complex, holistic, and culturally-embedded phenomenon that is shaped by a variety of social, cultural, and historical factors.

In the context of learning code, activity theory provides a useful perspective on the process of learning to program. According to activity theory, learning is not simply a matter of acquiring new knowledge and skills, but rather is a process of transforming existing practices and cultural tools. This means that in order for individuals to learn to code, they need to engage in meaningful and relevant activities that are connected to the cultural practices and tools of the programming community.

In practice, this means that learners need to be involved in a supportive community of coders and developers, who can provide guidance, feedback, and mentorship as they develop their skills. It also means that learning activities need to be designed in a way that is connected to real-world projects and problems, rather than simply teaching abstract concepts in isolation.

One key aspect of activity theory that is particularly relevant for learning to code is the concept of “expansive learning.” This refers to the idea that learning is not just a matter of acquiring new skills, but also a process of transforming and expanding the individual’s existing practices and cultural tools. For example, as learners engage in coding activities and projects, they are not only learning new programming concepts and syntax, but they are also expanding their understanding of the cultural practices and norms of the programming community.

## Game Making Pedagogies and Tools

### Code Playgrounds

Code playgrounds are online tools that allow users to experiment with and learn coding in a fun and interactive way. They are designed to be accessible and user-friendly, making it easier for beginners to get started with coding and for experienced coders to quickly test new ideas and concepts. The goal of code playgrounds is to provide a platform for users to learn and experiment with code without the barriers and distractions of a traditional development environment.

Code playgrounds typically provide a coding interface where users can write, test, and execute code snippets in a variety of programming languages. The user interface is often similar to that of a traditional text editor, with features such as syntax highlighting and auto-completion to make the coding experience more intuitive and efficient. Some code playgrounds also offer additional features such as a file system or project management tools, making it possible to save and organize code snippets and projects.

One of the key advantages of code playgrounds is that they provide a low-stakes environment for users to experiment with code. Since code playgrounds typically do not require any installations or setup, users can quickly and easily experiment with new code snippets without having to worry about the complexities of a full-fledged development environment. This can make code playgrounds a great way for beginners to get started with coding and learn the basics of programming.

Another advantage of code playgrounds is that they often include collaboration features, allowing users to share their code with others and work on projects together in real-time. This can be especially useful for students, teachers, and other users who are working on group projects or who want to learn from others. With the ability to share code and work on projects together, code playgrounds can also foster a sense of community and support among users.

There are several different types of code playgrounds available, each designed for a specific purpose or audience. Some code playgrounds, such as Codepen and JSFiddle, are designed for web developers and focus on front-end technologies such as HTML, CSS, and JavaScript. Others, such as Repl.it and Playcode, offer a more general-purpose coding environment and support a wider range of programming languages.

One of the challenges of code playgrounds is that they can be less powerful than traditional development environments. Since code playgrounds are designed to be easy to use and accessible, they often lack the advanced features and functionality of a full-fledged development environment. This can make it more difficult for users to perform complex tasks or to implement more advanced coding concepts.

### Learning Coding through Tinkering

Tinkering as a pedagogy is based on the idea that learning is an active, hands-on process that involves exploration and experimentation. This approach is often contrasted with more traditional, lecture-based approaches to education, which emphasize the transmission of information from teacher to student (Bers et al., 2014; Bevan et al., 2014, 2015; Gutwill et al., 2015; Vossoughi and Bevan, 2015). By providing opportunities for hands-on exploration and experimentation, tinkering helps learners to develop a deeper understanding of scientific and technological concepts.

In the context of coding, tinkering refers to the process of experimenting and playing with code to gain a deeper understanding of how programming works (Gutwill et al., 2015). This approach to learning coding provides a creative, engaging and fun way to learn, that can help individuals to acquire programming skills in a way that is accessible and meaningful to them. Tinkering with code involves experimenting with different codes, changing variables and testing the outcomes, which can help individuals to develop a deeper understanding of programming concepts. By tinkering, individuals can experiment with code without the pressure of producing a specific outcome, which can lead to a more relaxed and enjoyable learning experience.

Tinkering with code can also help individuals to develop their problem-solving and critical thinking skills, as they are encouraged to experiment, test and troubleshoot their code. This approach can help individuals to develop a more holistic understanding of programming, as they are able to understand the relationships between code and outcome, and learn how to apply programming concepts in real-world situations.

Another benefit of tinkering with code is that it allows individuals to learn at their own pace, and explore topics that interest them. Tinkering provides an environment where individuals can experiment and play, which can help them to develop their own projects and ideas, leading to a deeper sense of engagement and motivation.

### Remixing Games as a Way of Learning Coding

One way that individuals, particularly students, have been exposed to coding is through the process of remixing games. Remixing games refers to the process of taking existing games and modifying them to create new experiences, content, and features. This approach to learning coding has been proven to be an effective way to foster an interest in programming while providing hands-on experience.

The idea of remixing games as a way of learning coding is rooted in the constructivist theory of learning. According to constructivism, students learn best through hands-on experience and by constructing their own understanding of new concepts. When individuals remix games, they are given the opportunity to modify and manipulate the code to create new experiences, solve problems, and learn from their mistakes. This approach is much more engaging than simply reading about coding concepts or watching video tutorials.

Moreover, remixing games is a low-stakes environment for learning coding. Students can experiment and make mistakes without worrying about damaging any important systems or programs. This low-stakes environment is beneficial for students who may be intimidated by the thought of programming and encourages them to take risks and try new things. Furthermore, remixing games provides students with immediate feedback, allowing them to quickly see the impact of their changes and adjust accordingly.

There are several game engines and platforms that support remixing games, including Scratch, Kodu Game Lab, and Stencyl. Scratch, developed by the Lifelong Kindergarten Group at the MIT Media Lab, is a visual programming language that allows users to create animations, games, and interactive stories. Kodu Game Lab is a visual programming language specifically designed for creating games, while Stencyl is a game development platform that allows users to create games using a block-based programming language.

One of the benefits of using these platforms is that they make coding more accessible to individuals who have little to no prior experience in programming. For example, Scratch uses a drag-and-drop interface, which eliminates the need to type out lines of code. This makes it easier for students to focus on the concepts behind coding, rather than getting bogged down by syntax and technical details. Additionally, these platforms provide students with a variety of pre-built assets and code blocks, which they can use to build their own games and projects.

There are several studies that support the effectiveness of remixing games as a way of learning coding. A study conducted by the Games, Learning, and Society Group at the University of Wisconsin-Madison found that students who used Scratch to create their own games reported higher levels of engagement and motivation compared to students who were taught coding through traditional methods (Barr et al., 2011). Furthermore, a study conducted by the Technology, Education, and Design Group at Harvard Graduate School of Education found that students who used Scratch to create games and interactive stories demonstrated significant growth in their computational thinking skills (Resnick et al., 2009).

### Constructionist Approaches

Constructionist approaches are a learning theory that emphasizes the hands-on creation and collaboration of meaningful projects in a social context. This approach to learning, popularized by Seymour Papert, is rooted in the belief that people learn best by actively constructing knowledge and understanding through experiences and reflection.

In a constructionist learning environment, students engage in creative and collaborative activities such as designing and building prototypes, writing code, and creating digital media. Through these activities, students develop computational thinking skills, problem-solving skills, and a deeper understanding of complex concepts. The focus is not solely on learning the technical skills, but rather on how those skills can be used to solve real-world problems and create meaningful projects.

Constructionist approaches are particularly effective for learning programming and coding because they provide a hands-on, interactive, and exploratory experience for students. Instead of simply being taught programming concepts and syntax, students are encouraged to experiment with different programming languages and platforms, modify existing code, and create their own programs and games. This helps them to gain a deeper understanding of the underlying principles of programming and how they can be applied in real-world scenarios.

Furthermore, the social and collaborative aspect of constructionist approaches is critical to their success. By working in teams, students can build on each other’s ideas, share their knowledge, and learn from each other. This fosters a supportive and encouraging environment that helps to demystify complex technical concepts and overcome obstacles.

#### Microworlds

The concept of microworlds, which Papert introduced in his work, refers to simplified computer simulations or models that represent real-world scenarios or phenomena. Microworlds provide a safe and controlled environment in which students can explore and experiment with concepts in a concrete and meaningful way (Papert, 1980, 1993; Harel and Papert, 1991).

Papert believed that microworlds were an effective tool for promoting computational thinking, which he defined as the ability to think logically, systematically, and recursively. He argued that microworlds could help students develop computational thinking skills by providing them with opportunities to experiment with computational processes and to reflect on their own thinking.

In addition to promoting computational thinking, Papert believed that microworlds could also help students develop a deeper understanding of the underlying concepts in STEM subjects such as mathematics, science, and engineering. He argued that by working with microworlds, students could engage in hands-on and minds-on learning, which would help them to develop a deeper and more meaningful understanding of the concepts they were studying.

The work of Papert and the concept of microworlds continue to be influential in the field of educational technology (Kafai and Resnick, 1996). Today, microworlds are used in a variety of educational settings, from primary schools to universities, and have been shown to be an effective tool for promoting computational thinking and improving student learning outcomes.

### Universal Design for Learning (UDL)

Universal Design for Learning (UDL) is a framework for designing learning environments that are accessible and usable for all learners, regardless of their abilities or disabilities(CAST, n.d.). UDL is based on the idea that all learners have different needs, preferences, and ways of learning, and that learning environments should be designed to accommodate these differences. In the context of STEM education, UDL is a valuable approach that can help to make STEM learning accessible and engaging for all students, including those with disabilities or special needs.

UDL in STEM education involves designing learning environments that offer multiple ways to access, engage with, and demonstrate understanding of STEM concepts and skills. For example, UDL in STEM might involve providing students with multiple means of representation, such as text, graphics, and audio, in order to support different learning styles. UDL might also involve providing students with multiple means of engagement, such as hands-on activities, simulations, and games, in order to keep all students engaged and motivated.

UDL is based on the idea that all learners have different learning needs and preferences, and that learning environments should be designed to accommodate these differences. This is particularly important in STEM education, where many students may struggle with certain concepts or skills due to differences in their learning styles or abilities. By using UDL principles to design STEM learning environments, educators can help to ensure that all students have equal opportunities to succeed in STEM, regardless of their abilities or disabilities.

UDL also has the potential to support equity and inclusion in STEM education. By providing multiple ways to access and engage with STEM concepts and skills, UDL can help to reduce barriers to STEM learning for students with disabilities or special needs. This can help to increase diversity and inclusiveness in STEM, and to increase opportunities for all students to succeed in these fields. One area of UDL that teachers can implement straightforwardly is to represent concepts in the classroom in a diversity of ways. In a related study, researchers Cook and colleagues (Cook et al., 2016) explored the alignment of UDL with another framework, CRA, which consists of a three stage model to support learners to develop concepts (Fyfe et al., 2014).

### Computational Thinking Patterns and Scalable Game Design

Computational thinking patterns are an important aspect of computer science education, and are increasingly being recognized as a valuable tool for teaching problem-solving and critical thinking skills. One prominent approach to teaching computational thinking patterns is based on the work of Repenning, who has developed a framework, scalable game design, for teaching these concepts in a way that is accessible and engaging for students of all ages and backgrounds(Repenning and Ioannidou, 2008; Basawapatna et al., 2010, 2011; Lamprou and Repenning, 2018). Scalable game design is a method for teaching computational thinking patterns that emphasizes the development of games as a means of learning. This approach is designed to be both engaging and accessible to a wide range of learners, regardless of prior experience or background. It is based on the idea that game design can be used as a way of teaching computational thinking and programming concepts in a way that is fun, interactive, and relevant to students.

One of the key features of Repenning’s approach is that it provides a visual representation of computational thinking patterns, which can help students to understand these concepts more easily. For example, students can work with visual models of algorithms, which can help to develop an understanding of how algorithms work, and how they can be used to solve problems. This can be especially beneficial for students who are not naturally inclined towards computer science or technology, as it provides a context for learning that is more engaging and relevant to their interests.

One of the key benefits of scalable game design is that it can help to develop a range of technical and creative skills. By working on game design projects, students can learn how to program, design and test game mechanics, and develop their ability to solve problems and think creatively. Additionally, scalable game design provides a way for students to learn about game design and development from a practical perspective. Students can work on projects that are both challenging and engaging, and can learn about the design and development process from start to finish. This can be an important experience for students who are interested in pursuing careers in game design or development, as it provides a hands-on understanding of what is involved in these fields.

Scalable game design is also highly flexible and can be adapted to suit a wide range of learning environments. It can be used in the classroom, in after-school programs, and even in online learning environments. This makes it an accessible and effective method of teaching computer science and technology to a wide range of learners.

### Game Design Patterns

Design patterns are reusable solutions to common problems that arise in software design. They provide a standardized and proven approach to solving these problems, which can help to improve the quality and efficiency of software development. Design patterns are often described as templates or blueprints for solving problems and are used by software developers to structure their code and ensure that it is maintainable, scalable, and efficient (Gamma et al., 1995; Dearden and Finlay, 2006).

Game design patterns are reusable templates or solutions that are used to solve common design problems in game development. These patterns provide a structure and a language for designers to communicate their ideas and solutions to each other, and they can help guide the design process and make it more efficient. Some of the most common game design patterns include mechanics, dynamics, and aesthetics patterns (Schell, 2008; Hunicke et al., n.d.).

Mechanics patterns describe the core rules and systems that govern gameplay. For example, a mechanic pattern might describe how players collect and use resources in a game, or how they progress through levels and challenges. Mechanics patterns are the building blocks of game design, and they provide a foundation for the rest of the design process (Bjork and Holopainen, 2005; Björk and Holopainen, 2006).

Dynamics patterns describe the relationships and interactions between mechanics, and how they create emergent behaviours and challenges for players. For example, a dynamics pattern might describe how players compete or cooperate with each other, or how they encounter and overcome obstacles in a game. Dynamics patterns help to shape the overall experience of a game and make it more engaging and dynamic.

Aesthetics patterns describe the emotional, sensory, and narrative elements of a game that contribute to its overall style and feel. For example, an aesthetics pattern might describe how a game uses music and sound effects to create a specific atmosphere, or how it presents a story and characters to players. Aesthetics patterns play a crucial role in shaping the player’s experience and making a game memorable and enjoyable (Hunicke et al., n.d.).

### Game Design Patterns as a way to learn Game Making

Game Design Patterns provide a useful framework for novice game developers to learn game making and programming. They offer a common language and structure for designing and creating games, making it easier for new programmers to understand and implement complex game mechanics. By studying and applying game design patterns, novice game developers can learn the fundamental concepts of game development, including game mechanics, dynamics, and aesthetics, and use these concepts to create their own games (Repenning et al., 2010; Eriksson et al., 2019).

One of the advantages of using game design patterns is that they provide a set of reusable templates that can be adapted and customized to meet the needs of different games. For example, a novice game developer can use a pattern for collecting resources in a game and modify it to create a unique game mechanic that is tailored to the specific needs of their game. This allows novice game developers to quickly create engaging and well-designed games without having to start from scratch (Eriksson et al., 2019).

Another advantage of game design patterns is that they can help novice game developers understand the design process and make it more efficient. By using game design patterns, novice game developers can focus on the creative and technical aspects of game development, and avoid spending time on repetitive tasks and implementing basic mechanics. This allows them to concentrate on developing innovative and unique games that are engaging and enjoyable for players (Eriksson et al., 2019).

### The “Use, Modify, Create” (UMC) approach

The “Use, Modify, Create” (UMC) approach is a pedagogical approach to teaching coding that emphasizes hands-on learning through experimentation and exploration (Lee et al., 2011; Sentance and Waite, 2017; Lytle et al., 2019; Martin et al., 2020). The UMC approach is based on the idea that the best way for students to learn coding is by actively engaging with it, rather than simply being lectured on its concepts. This approach encourages students to start with existing code, games, and programs, and then modify and build upon them to create their own unique creations.

UMC is often used in computer science and technology education as a way to engage students in the learning process and provide them with meaningful, hands-on experiences. For example, students may start by modifying an existing game or program, such as changing the colours, adding new features, or altering the gameplay mechanics. This process of modification allows students to develop a deeper understanding of how the code works, as well as build their own skills and confidence as coders (Lytle et al., 2019).

Once students have gained experience through modification, they are then encouraged to create their own games and programs from scratch. This often involves working in small groups or teams, which fosters collaboration and encourages students to share ideas and support each other in the creation process. Through this hands-on, experiential learning approach, students are able to apply what they have learned and develop a deeper understanding of coding concepts and techniques (Sentance and Waite, 2017).

UMC is also used in game design and development education, as a way to foster creative thinking and innovation. By allowing students to modify and create their own games, they are given the opportunity to express their unique ideas and perspectives, and to develop a personal connection to the content they are learning (Lee et al., 2011).

## Game Making Communities

### Leveraging Funds on Knowledge to create Game Making communities

The concept of “funds of knowledge” refers to the array of skills, knowledge, and resources that exist within a community (Moll et al., 1992; Moje et al., 2004; Moll, 2019). This can include practical skills such as cooking or carpentry, as well as cultural traditions and values that shape a community’s identity. Developing communities using these funds of knowledge can be an effective way to empower communities and build social capital.

One of the key ways that funds of knowledge can be leveraged is by creating opportunities for community members to share their skills and knowledge with others (Moje et al., 2004). This can take the form of classes or workshops, where community members can teach others what they know. This can create a sense of pride and ownership in the community, as people recognize the value of what they have to offer. Additionally, this type of sharing can foster a sense of community as people come together to learn and grow.

Developing communities for young learners using funds of knowledge can be a valuable approach for creating inclusive and empowering learning environments. This approach recognizes that young learners bring a wealth of skills, knowledge, and resources to the learning process, and seeks to tap into these funds of knowledge to create a more meaningful and engaging learning experience.

One of the key ways that funds of knowledge can be leveraged in the development of communities for young learners is in the development of communities for young learners is to incorporate real-world, hands-on learning experiences into the curriculum (Moll, 2019). For example, if a community has a strong tradition of farming, this could be used to create gardening or agriculture projects for students. Similarly, if a community has a strong tradition of crafting, this could be incorporated into art or design projects for students. These hands-on learning experiences can provide students with a sense of purpose and relevance, as they are able to see how the skills and knowledge they are learning can be applied in the real world.

### The Development of Game Making Communities: Industry-based Resources and Initiatives

Game making communities are a vital part of the game development industry. These communities provide game makers with a supportive and collaborative environment where they can share their work, ask questions, and receive feedback from others in the community. Over the years, several resources and initiatives have been developed to support the growth and development of game making communities.

One of the primary resources for game making communities is online forums and communities. These platforms provide game makers with a space to share their work, ask questions, and receive feedback from others in the game development community. Some of the most popular online forums and communities for game makers include Reddit, itch.io, and Game Jolt. These platforms offer a wealth of information and resources on game development, and provide game makers with a supportive community of peers to connect with.

In addition to online forums and communities, there are numerous game development resources and tutorials that are available to game makers. These resources can be found on platforms such as Udemy, Coursera, and YouTube, and cover a wide range of topics, from game design to programming and marketing. These resources can be especially helpful for novice game makers who are looking to learn the basics of game development.

Another resource that supports the development of game making communities is game development conferences and events. These events provide game makers with the opportunity to network with other game developers, attend workshops and talks, and learn about the latest developments in the game development industry. Some of the most well-known game development conferences and events include the Game Developers Conference (GDC), the Independent Games Festival (IGF), and the Global Game Jam.

In addition to these resources, there are several initiatives that have been developed to support the growth and development of game making communities. One such initiative is the Global Game Jam, an annual event that brings together game makers from around the world to collaborate and create new games over the course of a weekend. The Global Game Jam provides game makers with an opportunity to work with other game developers, and to showcase their work to a wider audience.

Another initiative that supports the development of game making communities is the Indie Game Developer Network (IGDN). The IGDN is a non-profit organization that provides game developers with the resources and support they need to develop their games. The organization offers a range of services, including funding opportunities, mentorship programs, and access to industry resources.

Finally, game development grants and funding initiatives are another important resource that supports the development of game making communities. These grants and funding initiatives provide game makers with the financial resources they need to develop their games, and help to support the growth and development of the game development industry as a whole. Some of the most well-known grants and funding initiatives for game makers include the Indie Fund, the Games for Change Fellowship, and the Intel Game Dev Grant.

### Scratch - An On-line Community for Young Coders

Scratch is a popular online community that provides a platform for young people to create, share, and collaborate on interactive games, animations, and stories. The community is based on Scratch, a graphical programming language and software developed by the Lifelong Kindergarten Group at the Massachusetts Institute of Technology (MIT).

Scratch provides an intuitive, block-based programming interface that makes it easy for young people to create and share interactive projects. The community has a strong focus on collaboration, with users able to share their projects, comment on each other’s work, and remix each other’s projects to create new and original content.

The Scratch online community has become a vibrant and dynamic space for young people to express themselves and learn about computer science and digital media. The community has a wide range of projects and activities, including games, animations, stories, and simulations, that are created by young people from all over the world. The community also provides a range of educational resources and tutorials, including online forums and discussion groups, that help users to learn new skills and get feedback on their work.

The Scratch community has been shown to have a number of positive impacts on young people, including increased engagement and motivation in computer science, improved problem-solving and funds of knowledge critical thinking skills, and enhanced creativity and collaboration. It has also been shown to be a valuable tool for promoting diversity and inclusion in technology, as it provides a platform for young people from diverse backgrounds to express themselves and showcase their skills.

## Conclusion

In conclusion, game making can offer numerous benefits as an educational process. Firstly, it can provide a hands-on and interactive learning experience for students, allowing them to engage with and apply their knowledge in a practical and creative manner. Additionally, game making can foster critical thinking, problem solving and collaboration skills as students work together to conceptualize, design, and create their games. It can also promote digital literacy, as students learn how to use technology in new and innovative ways. Furthermore, game making can build self-esteem and confidence, as students are given the opportunity to showcase their skills and creativity to a wider audience.

Furthermore, game making can also play a role in preparing students for the 21st-century workforce by developing their technical and digital skills. With the increasing demand for technology-literate individuals in the job market, the skills and knowledge gained through game making can provide students with a competitive edge.

Allan, W., Coulter, B., Denner, J., Erickson, J., Lee, I., Malyn-Smith, J. and Martin, F. (2010) ‘Computational thinking for youth.’ *White Paper for the ITEST Small Working Group on Computational Thinking (CT)*.

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C. and Marshall, K. S. (2011) ‘Recognizing Computational Thinking Patterns.’ *In* *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM (SIGCSE ’11), pp. 245–250.

Basawapatna, A. R., Koh, K. H. and Repenning, A. (2010) ‘Using scalable game design to teach computer science from middle school to graduate school.’ *In* *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education - ITiCSE ’10*. Bilkent, Ankara, Turkey: ACM Press, p. 224.

Bers, M. U., Flannery, L., Kazakoff, E. R. and Sullivan, A. (2014) ‘Computational thinking and tinkering: Exploration of an early childhood robotics curriculum.’ *Computers & Education*, 72, March, pp. 145–157.

Bevan, B., Gutwill, J. P., Petrich, M. and Wilkinson, K. (2015) ‘Learning Through STEM-Rich Tinkering: Findings From a Jointly Negotiated Research Project Taken Up in Practice.’ *Science Education*, 99(1) pp. 98–120.

Bevan, B., Petrich, M. and Wilkinson, K. (2014) ‘Tinkering is serious play.’ *Educational Leadership*, 72(4) pp. 28–33.

Bjork, S. and Holopainen, J. (2005) *Patterns in game design*. 1st ed, Hingham, Mass: Charles River Media (Charles River Media game development series).

Björk, S. and Holopainen, J. (2006) ‘Games Design Patterns.’ *The game design reader* pp. 410–437.

Brennan, K. and Resnick, M. (2012) ‘New frameworks for studying and assessing the development of computational thinking’ p. 25.

CAST (n.d.) *About Universal Design for Learning*. CAST. [Online] [Accessed on 13th February 2022] https://www.cast.org/impact/universal-design-for-learning-udl.

Cook, S., Rao, K. and Cook, B. (2016) ‘Using universal design for learning to personalize an evidence-based practice for students with disabilities.’ *In*, pp. 239–248.

Curzon, P., Dorling, M., Ng, T., Selby, C. and Woollard, J. (2014) ‘Developing computational thinking in the classroom: a framework.’

Dearden, A. and Finlay, J. (2006) ‘Pattern Languages in HCI: A Critical Review.’ *Human–Computer Interaction*, 21(1) pp. 49–102.

Eriksson, E., Baykal, G. E., Björk, S. and Torgersson, O. (2019) ‘Using Gameplay Design Patterns with Children in the Redesign of a Collaborative Co-located Game.’ *In* *Proceedings of the 18th ACM International Conference on Interaction Design and Children*. Boise ID USA: ACM, pp. 15–25.

European Centre for the Development of Vocational Training (ed.) (2009) *European guidelines for validating non-formal and informal learning*. Luxembourg: Office for Official Publications of the European Union.

Fyfe, E. R., McNeil, N. M., Son, J. Y. and Goldstone, R. L. (2014) ‘Concreteness Fading in Mathematics and Science Instruction: a Systematic Review.’ *Educational Psychology Review*, 26(1) pp. 9–25.

Gamma, E., Johnson, R., Vlissides, J. and Helm, R. (1995) *Design patterns: elements of reusable object-oriented software*. Addison-Wesley.

Grover, S. and Pea, R. (2017) ‘Computational Thinking: A Competency Whose Time Has Come.’ *In*.

Gutwill, J. P., Hido, N. and Sindorf, L. (2015) ‘Research to practice: Observing learning in tinkering activities.’ *Curator: The Museum Journal*, 58(2) pp. 151–168.

Harel, I. and Papert, S. (eds) (1991) *Constructionism: research reports and essays, 1985-1990*. Norwood, N.J: Ablex Pub. Corp.

Hunicke, R., LeBlanc, M. and Zubek, R. (n.d.) ‘MDA: A Formal Approach to Game Design and Game Research’ p. 5.

Kafai, Y. B. (1994) *Minds in Play: Computer Game Design As A Context for Children’s Learning: Computer Game Designs as a Context for Children’s Learning*. 1 edition, Hillsdale, N.J: Routledge.

Kafai, Y. B. and Burke, Q. (2017) ‘Computational Participation: Teaching Kids to Create and Connect Through Code.’ *In* *Emerging Research, Practice, and Policy on Computational Thinking*. Springer, Cham (Educational Communications and Technology: Issues and Innovations), pp. 393–405.

Kafai, Y. B. and Resnick, M. (1996) *Constructionism in Practice: Designing, Thinking, and Learning in A Digital World*. 1 edition, Mahwah, N.J: Routledge.

Kafai, Y. and Burke, Q. (2014) ‘A decade of programming games for learning: From tools to communities.’ *In* Aguis, H. and Angelides, M. C. (eds) *Handbook of Digital Games*. NY: Wiley.

Kafai, Y. and Burke, Q. (2015) ‘Constructionist gaming: understanding the benefits of making games for learning.’ *Educational Psychologist*, 50(4) pp. 313–334.

Lamprou, A. and Repenning, A. (2018) ‘Teaching how to teach computational thinking.’ *In* *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: Association for Computing Machinery (ITiCSE 2018), pp. 69–74.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. and Werner, L. (2011) ‘Computational thinking for youth in practice.’ *ACM Inroads*, 2(1) pp. 32–37.

Lye, S. Y. and Koh, J. H. L. (2014) ‘Review on teaching and learning of computational thinking through programming: What is next for K-12?’ *Computers in Human Behavior*, 41, December, pp. 51–61.

Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., Isvik, A., Bounajim, D., Wiebe, E. and Barnes, T. (2019) ‘Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes.’ *In* *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. Aberdeen Scotland Uk: ACM, pp. 395–401.

Martin, F., Lee, I., Lytle, N., Sentance, S. and Lao, N. (2020) ‘Extending and Evaluating the Use-Modify-Create Progression for Engaging Youth in Computational Thinking.’ *In* *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. Portland OR USA: ACM, pp. 807–808.

Mercer, S. (2012) ‘The complexity of learner agency.’ *Apples-Journal of Applied Language Studies*.

Moje, E. B., Ciechanowski, K. M., Kramer, K., Ellis, L., Carrillo, R. and Collazo, T. (2004) ‘Working toward third space in content area literacy: An examination of everyday funds of knowledge and Discourse.’ *Reading Research Quarterly*, 39(1) pp. 38–70.

Moll, L. C. (2019) ‘Elaborating Funds of Knowledge: Community-Oriented Practices in International Contexts.’ *Literacy Research: Theory, Method, and Practice*. SAGE Publications Inc, 68(1) pp. 130–138.

Moll, L. C., Amanti, C., Neff, D. and Gonzalez, N. (1992) ‘Funds of knowledge for teaching: using a qualitative approach to connect homes and classrooms.’ *Theory into Practice*, 31(2,) pp. 132–141.

Papert, S. (1980) *Mindstorms: children, computers, and powerful ideas*. New York: Basic Books.

Papert, S. (1993) *The children’s machine: rethinking school in the age of the computer*. New York: BasicBooks.

Patrick, W. (2010) *Recognising Non-Formal and Informal Learning Outcomes, Policies and Practices: Outcomes, Policies and Practices*. OECD Publishing.

Repenning, A. and Ioannidou, A. (2008) ‘Broadening Participation through Scalable Game Design’ p. 5.

Repenning, A., Webb, D. and Ioannidou, A. (2010) ‘Scalable game design and the development of a checklist for getting computational thinking into public schools.’ *In* *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE ’10*. Milwaukee, Wisconsin, USA: ACM Press, p. 265.

Rogers, A. (2014) *The base of the iceberg: informal learning and its impact on formal and non-formal learning*. Leverkusen Opladen, Germany: Barbara Budrich Publishers (Study guides in adult education).

Sannino, A., Engeström, Y. and Lemos, M. (2016) ‘Formative Interventions for Expansive Learning and Transformative Agency.’ *Journal of the Learning Sciences*, 25(4) pp. 599–633.

Schell, J. (2008) *The Art of Game Design: A book of lenses*. 1 edition, Amsterdam ; Boston: CRC Press.

Sefton-Green, J. (2013) *Mapping digital makers: a review exploring everyday creativity, learning lives and the digital*. [Online] [Accessed on 18th December 2018] http://www.nominettrust.org.uk/.

Sefton-Green, J. and Erstad, O. (2012) ‘Identity, Community, and Learning Lives in the Digital Age.’ *In* Erstad, O. and Sefton-Green, J. (eds) *Identity, Community, and Learning Lives in the Digital Age*. 1st ed., Cambridge University Press, pp. 1–20.

Sentance, S. and Waite, J. (2017) ‘PRIMM: Exploring pedagogical approaches for teaching text-based programming in school.’ *In* *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*. Nijmegen Netherlands: ACM, pp. 113–114.

Toohey, K. and Norton, B. (2003) ‘Learner Autonomy as Agency in Sociocultural Settings.’ *In* Palfreyman, D. and Smith, R. C. (eds) *Learner Autonomy across Cultures*. London: Palgrave Macmillan UK, pp. 58–72.

Vossoughi, S. and Bevan, B. (2015) ‘Making and Tinkering: A Review of the Literature.’

Werquin, P. (2009) ‘Recognition of Non-formal and Informal Learning in OECD Countries: an Overview of Some Key Issues.’ *Recognition of Non-formal and Informal Learning in OECD Countries: an Overview of Some Key Issues*, (03/2009) March.

Wing, J. M. (2008) ‘Computational thinking and thinking about computing.’ *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881) pp. 3717–3725.