KCSB: A Game Construction Framework for Kids Coding in Taiwan

Chun-Hsiung Tseng, Yung-Hui Chen, Shin-Chi Lai, Jia-Rou Lin
Department of Communications, Yuan Ze University

Abstract: In this research, we proposed a game construction framework for designing games to teach young children programming in Taiwan. We focused on kids around 6-7 years old. For kids at such ages, solutions such as LEGO or Scratch may not be acceptable (considering that in Taiwan there is no official curriculum for teaching programming or computational thinking for kids at such ages). Although the blueprint setup a goal of information education for elementary schools, no real time slot is allocated in the current curriculum of elementary schools. The fact shows that teaching programming for young kids in Taiwan is difficult with existing tools/products. In this research, we proposed KCSB, a game construction framework which includes a set of runtime objects, a gd file parser, and a game creator. The framework is based on the Godot game engine. Designers can use the proposed framework with the Godot game engine to develop games that are suitable for teaching programming concepts for young children. At the current stage, the framework supports these programming concepts: problem solving, algorithm evaluation, function (reusable component) definition, and loop. In the manuscript, an example demonstrating the use of the framework is also included.

Keywords: KidsCoding; Programming

1. Motivation

The application domain of programming skills has become wider and wider. It is usually considered that starting learning programming in an earlier stage of life will result in better progress. There are quite a few kids coding related activities around the world. For example, Tynker offers several programming learning modules for kids beyond 7 years old\(^1\). Tynker offers several game modules, code robots, and apps for the STEM purpose. In Eesti, kids have to attend programming courses when they attend elementary schools. They usually use products such as Kodu Game Lab, Logo MSW, Scratch, and LEGO Mindstorms. KidsCodingTW\(^2\) is an organization in Taiwan and provide python course for junior high school (or above) students. Focusing on those targeting at kids around 6-7 years old (first degree of elementary school or kindergarten students in Taiwan), it appears that most solutions adopt the game-based learning methodology. For kids at such an age it is really difficult to setup education goals for programming. Existing products, such as Scratch and LEGO Mindstorms are typically designed for kids beyond 7 years old (Scratch is designed for 8-16 years old\(^3\); LEGO Mindstorms is for 9+ years old\(^4\)). The Ministry of Education in Taiwan proposed a blueprint for information education in 2016. Although the blueprint setup a goal of information education for elementary schools, no real time slot is allocated in the current curriculum of elementary schools. The fact shows that teaching programming for young kids in Taiwan is difficult with existing tools/products. An additional issue for existing tools/products is that there is no much room left for teachers to adjust their teaching curriculum. For kids below 7 years old, there are usually fixed games delivering fixed concepts.

Copyright © 2018 Chun-Hsiung Tseng et al.
doi: 10.18063/bdci.v2i1.741
This is an open-access article distributed under the terms of the Creative Commons Attribution Unported License (http://creativecommons.org/licenses/by-nc/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
To address these issues, we developed this framework. The design of framework has the following goals in mind:

1. It is aimed at teaching programming concepts for kids below 7 years old
2. With the proposed framework, teachers can easily construct a customized game
3. Despite of its simplicity, it can still be used for constructing games with good quality

2. Related Works

In recent years, the capability to do programming is considered as an important skill. According to Duncan, C., Bell, T., & Tanimoto, "There has been considerable interest in teaching "coding" to primary school aged students, and many creative "Initial Learning Environments" (ILEs) have been released to encourage this."(Duncan, Bell, & Tanimoto 2014) At which age should programming concept been taught remains a controversial issue, but, according to their research, allowing kids to explore programming concepts is important. Which programming concepts should be taught for kids? The research of Martinez, Gomez, & Benott gives us some clues. According to their research results, "Data show that all students can intuitively learn sequence, conditional, loops and parameters." The research targeted kids at 3-11 years old and they stated that "These findings imply that introducing CS in mandatory schooling from an inquiry based approach is both achievable and beneficial." (Martinez, Gomez, & Benott 2015) The proposed platform thus reasonably focused on sequence and loops. But what are the proper methods for teaching kids programming concepts? Based on the survey made by Sáez-López., Román-Gonzálezed, & Vázquez-Cano, "visual programming" may be a good approach (Sáez-López., Román-González, & Vázquez-Cano 2016). Their study assessed the use of a Visual Programming Language using Scratch in classroom practice and targeted students at the 5th and the 6th grade. Aivaloglou and Hermans pointed out that "Block-based programming languages like Scratch, Alice and Blockly are becoming increasingly common as introductory languages in programming education." (Aivaloglou & Hermans 2016) Their survey showed that there is substantial research showing that these visual programming environments are suitable for teaching programming concepts. They developed a method to detect code smells for codes built with such type of environments and thus can assess how well kids program. Gamification is also considered useful. According to Prensky, games is a good opportunity to get kids involved in programming (Prensky 2003). His research focuses on video games but other types of games should demonstrate similar characteristics. Werner, Denner, & Campe also proposed that programming games are beneficial for assessing the computation learning level of kids (Werner, Denner, & Campe 2015). Perhaps what helps most is not the "game" itself but the "play" behavior. Corneliusen and Prøtz surveyed voluntary groups teaching children and youth basic computer coding skill and draw the conclusion that "The activities emphasize play, while teaching principles of computer science." (Corneliusen & Prøtz 2016) Other case studies also concluded that "play" is a important factor to help kids learn programming. The research of Sullivan, Bers, & Mihm proposed the use of KI-BO, which is a tool set allowing youngchildren to become engineers by constructing robots using motors, sensors, and craft materials (Sullivan, Bers, & Mihm 2017). They concentrated on teaching computational thinking, which is a central concept of programming. The experiment held by Strawhacker and Bers adopted a similar approach but they used the LEGO WeDo robotics (Strawhacker & Bers 2015). Fessakis, Gouli, and Mavroudi used a Logo-based environment on an Interactive White Board (Fessakis, Gouli, & Mavroudi 2013).

Most, except few existing researches were aimed at kids older than 8 years old. Younger children, kids before 7 years old, behave pretty different. Should we introduce digital game-based learning to younger kids is a even more controversial question. The research of Price, Jewitt, and Crescenzi discussed the effect of iPads in pre-school children’s mark making development (Price, Jewitt, & Crescenzi 2015). According to their experiment results, in that way, kids can do more mark making and the results are not completely bad. The hardware tool is one thing, the software tool is another. Papadaki, Kalogiannakis, and Zaranis investigated the effect of using Scratch Jr, which is a special version of Scratch and is designed for younger kids, to teach programming concepts to preschool children (Papadaki, Kalogiannakis, & Zaranis 2016). As the researchers stated: “the teaching of programming and development of fundamental programming concepts at the preschool age has attracted the interest of the educational and scientific community.” They performed a small-scale pilot study for the evaluation of Scratch Jr and the result was good. Furthermore, Manches and Plowman pointed out that although there has been a proliferation of programming tools designed for kids, the pedagogy
to be used is still the most mattered (Manches & Plowman 2017).

3. The Framework

The framework consists of three parts: the runtime components, a gd file parser, and a game creator. The runtime components are a set of gd files to be used with the Godot runtime. They bridge the framework with Godot objects. The gd file parser will parse existing gd files created with the Godot game engine and extract the needed parts to be used as the input of the game creator. The game creator is a editing tool help designer use the framework. The figure below illustrates these components:

Figure 1: Components of the framework.

3.1 Definition of Objects

The framework is designed to be used with the Godot game engine. To keep simplicity, we focus on maze-style games. Such a type of game is easier to be understood by kids below 7 years old and it is also easier to embed programming concepts such as looping and function defining in the resulting games. The framework includes four sets of modules: the tile map, the player (the main character), objects on the map, and utilities. These objects will be explained below:

1. The tile map: the map itself; within the framework, it is assumed that there is only a single map for each level; note that the size of the map is not limited; therefore, even though there is only one tile map object for one level, a tile map can visually across several screens; all map objects within a level, including the main character, are maintained by the single tile map object

2. The player: the major character on the map; it can be directly controlled by users (via gestures or button clicks) or driven by some commands; the player is just a special type of map objects and thus is also maintained by the belonging tile map

3. Map objects: monsters, items, and obstacles, etc. are all possible types of map objects; these objects can be either dynamically created according to events or just be put in the map at the initial phase

4. Utilities: utility modules to help developers build their games

The framework is to be used with the Godot game engine, and thus some fundamental building blocks of Godot deserves some explanations. A Godot object can be separated into two parts: the presentation layer, which is defined with a tscn file and the control layer, which is implemented via a gd file. The gd file usually defines a class inheriting from a built-in Godot class such as a TileMap, a KinematicBody2D, or a Node. To keep Godot objecting working properly, the associations between tscn files and gd files must be maintained. However, in order to implement the functionalities proposed earlier, we have to create a set of classes and enforces some class structures. To keep objects in both worlds work, the adapter pattern is used. The structure is shown below:
The proposed framework contains three gd files to implement the above modules: GridBase.gd, GridObject.gd, and Global.gd. First of all, GridBase.gd implements common tasks of a tile map. GridBase.gd contains the following methods:

1. initMap(): a callback method that will be invoked when the tile map is being initialized, a tile map implementation will usually override the method to implement the initialization procedure
2. spawnObject(gridX, gridY, gdPath, name): invoke the method to create an object and place it on the location specified by (gridX, gridY); gdPath is the path to the definition file of the object to be created while name is the name to be assigned to the object
3. getObjectFromNode(node): return the corresponding framework object for the given Godot object
4. addObject(gridX, gridY, gridObject): add the specified gridObject (a framework object) to the specified position
5. removeObject(gridObject): remove the specified framework object from the tile map
6. removeObject(gridX, gridY): remove all framework objects placed on the specified position of the map
7. getObjects(gridX, gridY): return all framework objects placed on the specified position of the map
8. getObjectByName(gridX, gridY, name): return the framework object with the specified name and is placed on the specified position
9. getFirstObject(gridX, gridY): return the first framework object placed on the specified position of the map
10. isGridCellVacant(gridX, gridY): return true if the specified cell is empty (with no framework object placed on it)
11. getWorldPos(gridX, gridY): convert the specified map-based position to screen-based position (pixels)
12. getMapPos(worldX, worldY): convert the specified screen-based position to map-based position
13. is_goal(pos): return true if the player object has reached the goal of this game; child class must override this method to set up its own logic

Secondly, GridObject.gd implements the common behaviors of a framework object. GridObject.gd contains the following methods:

1. init(_name, _type, _tscnPath): initialize a framework object; _name refers the unique name used for identifying the object; _type is the type or the category of the object; note that _name and _type are specified and given the meanings by the game designer; _tscnPath is the path to the corresponding tscn file
2. getNode(): return the Godot object associated with this framework object
3. getParentGrid(): return the container GridBase object of this framework object
4. isBlocking(): return true if this framework object should block other objects from entering this cell; implementations should override this method to define their own semantics; usually, for obstacles such as walls or rocks, this method should return true
5. getGridPos(): return the map-based position of this framework object
6. getWorldPos(): return the screen-based position of this framework object
7. isKinematicObject(): return true if this framework object can move

Global.gd is a utility class implementing some common game management tasks. The following methods are de-
fined:

8. reset(): reset all status of this game
9. gamepoint(mapid, stuid, gamenpoint, datetime, blocknum, complete): store the scores the user gained in this game
10. gamestatus(req): store the detailed status information of this game; req is a JSON string with its format specified by the game designer

3.2 Using the Framework

The framework is not a fully-fledged game engine itself. We utilized the class inheritance method of the Godot game engine. GridBase.gd extends Godot’s TileMap and thus can be used with the TileMap tscn node. The figure below shows a possible configuration (in the Godot game engine):

![Figure 3](image)

In the figure above, the node ”Grid” is a TileMap object. To activate the framework, change the signature of the auto-generated gd script from extends TileMap to

```gd
extends 'res://GridBase.gd'
```

Besides, for each object (usually extends from Godot’s Node2D or KinematicBody2D class), create a wrapper framework object. The wrapper should extends from GridObject.gd and serves as the connection between the proposed framework and the Godot game engine.

For each scene in the game, there should be a GridBase implementation. A simple implementation is shown below:

```gd
var map=[
    [10,5]:["res://cooker/Pot.gd", "Pot"],
    [10,1]:["res://cooker/Knife.gd", "Knife"],
    [8,1]:["res://cooker/Onion.gd", "Onion"],
    [8,5]:["res://cooker/Potato.gd", "Potato"],
    [3,3]:["res://cooker/Loop.gd", "Loop"],
    [12,6]:["res://cooker/Guest.gd", "Guest"],
    [7,6]:["res://cooker/One.gd", "One"],
    [6,1]:["res://cooker/Half.gd", "Half"],
]

func initMap():
    for entry in map:
        spawnObject(entry[0], entry[1], map[entry][0], map[entry][1])
    spawnObject(0, 6, "res://cooker/CookerPlayer.gd", "Player")
    return

func is_goal(pos):
    return true
```

![Figure 4](image)

In the example above, a map object is created to maintain the objects on the tile map. Then, the example overrides the initMap method:

```gd
func initMap():
    for entry in map:
        spawnObject(entry[0], entry[1], map[entry][0], map[entry][1])
    spawnObject(0, 6, "res://cooker/CookerPlayer.gd", "Player")
    return
```
spawnObject(0, 0, "res://cooker/CookerPlayer.gd", "Player")

In the above code snippets, it iterates all entries in the map, for each entry, the spawnObject method is invoked to create the corresponding framework object. Each entry is organized in the following way:

`[10,5]:["res://cooker/Pot.gd", "Pot"]`

[10,5] is the position of the resulting object on the map. res://cooker/Pot.gd is the path to the gd file containing the definition of the corresponding Godot object. Pot is the name of the resulting framework object. The definition of Pot.gd is shown below:

![Figure 5: The definition of Pot.gd, which is a framework object.](image)

The implementation above defines a init (name) method which in turn invokes the init() method defined in the GridObject class. The main player object of the game is defined in CookPlayer.gd and is shown below:

![Figure 6: The main player framework object.](image)

In the main player object, a tick(delta) method is defined to be compatible with Godot's original process mechanism. The code snippets implements the state machine below:
3.3 A Game Creator

To simplify the adoption of the framework, a game creator was also constructed. Currently, the creator supports only game projects based on the Godot game engine. The figure below shows the user interface of the creator:

Figure 8: The game creator.

On the left-hand side, a file tree is displayed. Users double click a gd file to activate its corresponding editor, which will be brought up and displayed on the right-hand side. The creator supports two types of gd files: the GridBase files and the GridObject files. The creator will automatically detect the base class of the selected gd files and choose the proper editor user interface. GridBase files define maps and the user interface of its editor is shown below:

Figure 7: The state machine of the example.

The Godot engine itself is very flexible and it supports various types of games. However, the flexibility becomes a problem with regard to kids coding. First of all, to make the delivery of target programming concepts effective, we want to make the "gaming" part lean. And second, in Taiwan, kids less than 7 years old (in kindergartens or the first degree of elementary schools) have a close relationship with their homeroom teacher. As a result, their homeroom teachers will understand them best. If teachers can directly design games and use the games in their courses, kids will feel interesting and good learning performance can be expected. To sum up, if we focus on kids coding for kids under 7 years old, a fully-fledged game engine may not be a best choice.
The GridBase editor hides the detail of the definition of a map. By entering the values of Grid Width, Grid Height, Tile Width, and Tile Height, the getGridSize method and the getTileSize method in the gd file will be automatically generated or updated. Users can click the Goal Function button, the Init Function button, or the Custom Codes button to bring up a text-based gd file editor to edit the corresponding parts of the GridBase file. Furthermore, users can click the Map Entries button to bring up the map editor, which provides a lean user interface for the construction of the map. The user interface of the map editor is shown below:

![Map Editor](image)

**Figure 10:** The map editor.

With the map editor, users simply specify which object will be displayed on which location of the map by entering values to the corresponding fields. With these editors, users typos can be prevented and the game creation process will be simpler.

If the selected file is a GridObject file, then the GridObject file editor will be displayed. There are two types of GridObjects: the static ones and the moving ones. The two types of GridObjects are distinguished by the kinematicObject field. For the static ones, the value should be false and vice versa. Besides, a GridObject has two mandatory fields: type and gdPath. The former specifies the type of the GridObject while the latter specifies the path to its corresponding Godot file. The GridObject file editor is shown below:

![GridObject Editor](image)

**Figure 9:** The GridBase editor.
3.4 Programming Concepts that Can be Implemented

The framework is shipped with some built-in facilities that can be used for teaching kids programming concepts. Note that developers can still implement their own modules to deliver other programming concepts. By default, the framework is most suitable for the following concepts:

1. Problem solving
2. Algorithm evaluation
3. Function (reusable component) definition
4. Loop

The figure below demonstrates a game created with the framework:

![Figure 11: The GridObject editor.](image1)

Kids are asked to lead the main character of the game to a pre-specified location by composing the given commands (the buttons on the right hand side). During the composition of commands, kids have to solve the given problem with the tools they have before they hit the ”START” button. This should help kids to think abstractly and therefore will help kids to develop problem-solving skills.

Additionally, the commands chosen will be saved in a accessible array, and thus can be used to evaluate how well the kids solve the problem. In order to achieve better result, kids have to evaluate their own solution. Therefore, this will enhance their ability of evaluating an algorithm.

The orange brackets button on the right hand side supports the creation of reusable commands. Kids use the opening bracket button to start a series of commands and use the closing bracket button to end the series. The whole steps are conceptually equivalent with creating the reusable function in formal programming languages.

Besides creating a reusable command set, kids use the loop button between the two bracket buttons to execute the set of commands. Kids will be tempted to solve the problem in this way since then a shorter command sequence can be
obtained. Under the hood, the framework supports the definition of reusable command sequences and will expand all references to the command sequences at run time.

4. Conclusions and Future Work

In this manuscript, we propose KCSB, which is a framework for creating games which are suitable for teaching programming concepts to young children under 7 years old. So far, the framework is in the prototyping stage, but it is usable and we have already used the framework to develop a set of kids coding games. These games have been used in an experiment for usability test. In the future, we set the following goals to accomplish:

1. Improve the internal gd file parser to be capable of processing more gd file syntax
2. Make the game creator even more user friendly
3. Arrange a larger scale experiment and analyze the collected results

References