Sitting Amongst Students to Learn Game Development with Microgamr

Flávio R. S. Coutinho
Departamento de Computação
CEFET-MG
Belo Horizonte, Brasil
fegemo@cefetmg.br

Glívia A. R. Barbosa
Departamento de Computação
CEFET-MG
Belo Horizonte, Brasil
glivia@cefetmg.br

Ismael S. Silva
Departamento de Computação
CEFET-MG
Belo Horizonte, Brasil
ismaelsantana@cefetmg.br

André R. da Cruz
Diretoria de Elétrica
IFSP
São Paulo, Brasil
dacruz@cefetmg.br

***Abstract*—This paper describes Microgamr: a learning object we created with the intent of aiding students to learn game development techniques collaboratively. The core concept is to turn the whole class, including the teacher, into a development team that will create a game whose challenge is to fulfill rapid challenges – called microgames – in succession. We applied Microgamr in two classrooms of a Digital Game Development discipline of a Computer Engineering graduation course, and it yielded two games. We could observe that the application of Microgamr allowed students to work together in an actively collaborative team and organically learn contents outside the scope of the class with a closer relationship with colleagues and teacher. It also exposed students to an environment similar to those found in game development soft-houses, in which medium to large teams work together in the same game.**

***Index Terms*—game development education, learning object, microgames**

I. Introduction

Game development involves expertise in multiple areas ranging from graphics, game design, music composition, and programming, usually requiring broad and multidisciplinary teams and a lot of time investment to be completed [1]. The inherent complexity of a game development project permeates difficulties in defining its scope, estimating deadlines and the eventual lack of resources, and it might lead to the project interruption, especially when the creators are smaller, less experienced or independent companies or teams [2]. It is a popular saying among independent game developers that to complete the final 10% of the project it usually takes 90% more of the planned schedule [2], which reinforces the need of careful planning.

Learning about game development presents similar challenges. There are graduate courses specific for the area and they contemplate formation in the whole spectrum of the involved areas [3], which enables the composition of multidisciplinary teams inside the same classroom. On the other hand, it is not uncommon for Computer Science-related graduate courses to offer isolated "game development" disciplines. As the students' profile in such courses is mostly programming-based, the assemble of multidisciplinary teams rarely happens, which reflects on the overall quality of the developed games, especially concerning art.

Still on the academic aspects of game development, the literature shows an extensive exploration of the use of games as learning tools for a multitude of areas [4], and also the use of game *development* as tools for learning Computer-related subjects [5], [6]. Curiously enough, not much work has been found regarding *teaching* game development. With the significant gap that exists between the experience of new graduates and the high (and usually unmet) expectation of the industry for them [3], [7], we stress on the necessity for improving the quality of the formation of students in the field.

In that context, this work aims at investigating the research question of *"how can we effectively teach/learn intricacies of game development, including some tacit knowledge, closer to how a game studio works, in a classroom?"* We conducted an applied research and proposed a learning object, with two applications as a preliminary evaluation that enabled us to generate insights into our research question.

In this sense, we present a *learning object* for a digital game development discipline situated in a Computer-related graduate course, which aims to enable the exercise of the whole creation process of a game, from its inception to its digital distribution, in a format that tries to resemble a soft-house. The learning object we propose is called **Microgamr** (pronounced as "microgamer") and it contemplates assembling the whole class, all students and teacher, to develop one single game. Such format provides several benefits over the alternative of splitting the class into small groups:

(I) it enables the creation of a game that is larger, more complex and more polished than one which would be developed by a small group of students with the same schedule;
(II) it improves the chances of having higher graphics/sound quality, as there is more diversity of students' profiles in a larger team;
(III) the participation of the teacher acting as the project manager ensures the delivery of the final product, while also allowing very close interaction with the students. Also with the role of a tech leader, the teacher has better opportunities to both aid in the students learning process as well as better inputs for evaluating them;
(IV) it exposes the students to an environment more similar to the one in the industry, reducing the gap between the

expectation of the students and the game companies;

(V) it provides a clear goal shared by all students and the teacher, improving the overall engagement, as the responsibility of fulfillment (delivering the game) is equally split among all participants;

To enable the concomitant work of the students in the same code base, it is necessary to use a version control system (VCS) and a hosting service to keep the project files available at all times, everywhere. We propose the use of a free VCS called Git [8] and the remote repository hosting service called GitHub [9]. Even though such kind of tool being of great importance in Software Engineering in general [10], [11] and being extremely popular in the development environments of companies and organizations, the Computer-related graduation courses in Brazil seldom include such topic in their curriculums. That can be partially explained by the absence of the subject in Brazilian Computer Society's reference curriculum for Computing courses [12].

We have applied Microgamr in two classes of a discipline called "Digital Games Development" in a Computer Engineering graduation course as a preliminary evaluation and the results showed high engagement of the students with the games developed, as well as their appreciation of the experience gathered by participating in the whole process of making a complete game.

The remainder of this work is organized such that Section II presents the notion of a learning object and briefly explains the concepts behind the tool and service used to allow concurrent work on the project; Section III introduces Microgamr, which is the learning object we proposed – its inspiration, details, phases and the technicalities of the project; Section IV describes our two experiences using it in the classroom and an evaluation of how it performed from the perspective of the students and the teacher; Section V describes Microgamr formaly as a learning object – how to find, adapt and use it; and Section VI highlights our conclusions and presents possible future works.

II. Theoretical Foundation

The main output of this work is a *learning object* called Microgamr. Hence, this section presents the notion of a *learning object* as an instructional tool to be (re)used in proper educational contexts. Later, this section presents some essential concepts around the software tools and services that are required to use Microgamr in a classroom environment.

A. Learning Objects

The notion of *learning object* in the literature can be traced back to 1967 [13], when Ralph Gerard digressed about the possibility of reusing "*small instructional blocks*" as learning tools which could be used as pieces to assemble more complex or longer educational courses. At such an early era of Computing and Networks, he envisioned the use of computers as a medium for cheap reproduction and distribution of such content, which could improve access to education while reducing expenditures with public schooling.

Later, David Wiley defined *learning object* as "small (relative to the size of an entire course) instructional components that can be reused many times in different learning contexts [14]" (page 4), but more generally as "any digital resource that can be reused to support learning." (page 6). They are characterized by their potential to be reusable, generic, adaptable and scalable to large numbers of audiences.

In 2002, IEEE standardized the metadata required to describe a learning object [15]. It comprises the learning object's (a) educational objective, (b) prerequisites for the learner to be able to use it, (c) the taxonomic topic, (d) the interaction model between the learner, instructor and object and the (e) technology requirements for executing it. The purpose of standardizing such information is to enable learning objects to be found, identified and reused [15].

B. Git and GitHub

Git is a version control system (VCS) [10], [11] created in 2005 by Linus Torvalds [8] with the initial purpose of managing the development of Linux's kernel source code. Leveraged by its great flexibility and performance, it quickly became the most used VCS world-wide [16]. Fig. 1 shows the evolution of the number of searches made by Internet users in the Google search engine for the most popular VCS's from January 2004 until 2019.

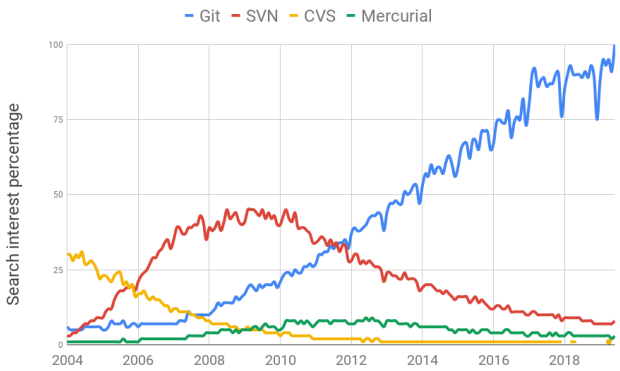

Fig. 1. Search popularity of the most popular VCS's on Google

The main goals of VCS tools contemplate (a) recording the changes made to the source code, (b) enabling multiple users to change the project files at the same time, (c) enabling and facilitating conflict resolution when multiple users simultaneously edit the same region of a file, and (d) enabling the navigation between different versions of the source code that have been recorded over time.

The directory which contains a software's source code is denominated a *Git repository*. The (a) record of the changes made to the repository files can be done through the creation of *commits*, which are the documentation of what has been changed, when and by whom. That enables changes to be tracked by creating points in history (d) to which the repository can return posteriorly, if necessary. When more than one user

changes the same file, Git attempts to (c) combine all the changes that are not in conflict with each other, i.e., those made at distinct lines of code. When that is not possible, the author of one of the commits must resolve the conflict with a *merge* operation (in Git terminology).

For the repository to be shared and used by more than one user (b), it requires a server to keep a copy of it (in the terminology, a remote repository) and it is desirable that it is made publicly available over the Internet at all times. Betting on the adoption of Git by open-source code developers, the company GitHub Inc [9] created, in 2007, a Git repository hosting service which is free of charge for projects that have public read permission (i.e., its code can be seen and downloaded by anyone).

In proposing and applying Microgamr twice so far, we used GitHub to host the remote repository of the games we developed so that the source code was available to all students and the teacher to work together on the project at all times and wherever necessary.

Besides the repository hosting service, GitHub also provides a Web interface that enables the visualization and management of the repository under various aspects. With it, it is possible for users without write permission in a repository to create a *fork* to themselves, which is a copy of the "original" repository where they have full permission to make the changes they wish, and, then, send them back to the original repository through an operation called *pull request*.

The owner of the original repository (in our case, the teacher) can then see exactly what has been changed by the student who opened the *pull request* and either accept it and integrate the changes to the original repository or evaluate and request changes to be made to the code before it is accepted.

Such workflow enables the teacher to evaluate the source code changes sent by students and to approve or request changes from them, keeping the code consistent, with high quality and usually error-free. The rejection (request for changes) of a *pull request* is also an opportunity for the teacher to show better ways of implementation, enforce coding best practices and to pinpoint code bad smells, giving students precious feedback and a chance to improve their knowledge and skills before being graded. Next, we present Microgamr.

III. Microgamr

The game to be developed by the whole class has to be easily split into tasks so they can be developed simultaneously by various students, who are possibly inexperienced in game development and VCS tools. Besides, it should have an adequate complexity level, but should also be flexible to comply with different levels of expertise and interest from the students.

We first describe the inspiration from which the Microgamr concept stems, then we present the game proposal that met our requirements as a learning object.

A. Inspiration: WarioWare

Every title in the WarioWare series puts the player in a challenge against a succession of *microgames*, which are games with only a single mechanic, usually unique among the other microgames, with simple controls and very short duration, of only a few seconds. Before each microgame, the player receives with a very brief instruction of what he/she should do – eg., *"Run!"*, *"Escape!"*, *"Don't let it fall!"*, and it is part of the challenge for the player to understand what he/she has to do and act promptly.

Fig. 2 depicts an example which shows the instruction "Finish Off" and the player needs to maneuver the Nintendo Wii controller as if it was a glass of water being taken to his/her mouth. If the player performs the movement too slowly, time runs out, but if he/she goes too quickly, the water spills on the character's face, and the player loses a life. The winning condition is triggered if the player manages to make the character drink all of the water in the available time (about 5 seconds) without spilling it on his/her face.

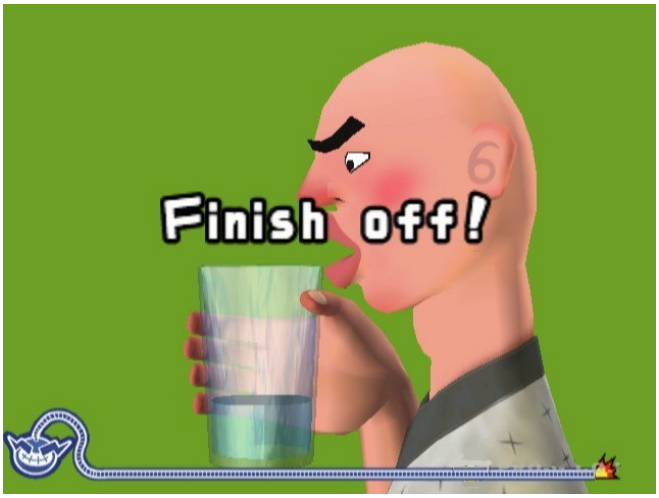

Fig. 2. A microgame from WarioWare: Smooth Moves [17]

As the winning condition of a microgame is satisfied, the game quickly changes to the next in the sequence, without the possibility to pause, and the next one is not necessarily related to the previous neither from the mechanics point of view nor its thematics. Indeed, there is no standardization of the microgames, which are deliberately distinct from one another both in their art style and in the challenging situation in which it puts the player.

Also inspired by WarioWare, the Australian company Metro Trains Melbourne, responsible for the suburban railway network in Melbourne, developed the series of games *Dumb Ways to Die* [18] comprised of 3 titles (2013, 2014 and 2017) for Android and iOS devices. They have the same format of *WarioWare*'s microgames, but they have the intention of alerting people of the dangers related to the lack of attention around railways and trains.

Those and other examples show that the WarioWare series inaugurated a genre (or a sub-genre of action/puzzle [1], depending on the taxonomy adopted) that is becoming more popular over time. Inspired by such game mechanics, we

propose the creation of one such game as practical work to learn game development, as presented next.

B. The Game Proposal

The game proposed for development by the class consisted of one with the same mechanics of those from the WarioWare series, based on sequences of microgames. There were two main reasons that drove our decision: (i) with a careful software architecture design, each microgame could be developed independently from each other, allowing students to work freely on their part, reducing the probability of *merge conflicts* to zero during this phase – which is desirable since resolving conflicts might not be easy for programmers that have little or no experience with VCS tools; and (ii) the deliberate lack of artistic consistency present in the WarioWare series among their microgames could be appropriated by us, alleviating the need to have a consistent art direction altogether, as that would be an arduous duty considering that the students are undergraduates in Computer-related degrees and might not have nice design skills.

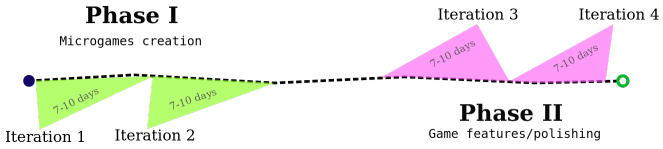

Fig. 3. Phases and iterations for the development of Microgamr

The project schedule was split into two phases with four iterations and managed in a SCRUM-ish scheme, with each iteration being 7-10 days long and having two status meetings per week (Fig. 3). In the first two iterations, which we called phase I, each pair of students developed two microgames, with the restriction that one of them should be a *"super-premium microgame"*, as it should fulfill one (or more) of the following criteria:

(I) use a physics engine in an interesting way
(II) use a tilemap as part of the scenario
(III) be in 3D instead of 2D
(IV) have some kind of intermediate level of artificial intelligence
(V) have two phases in the same game (like a "bonus" or "boss" second phase)
(VI) use audio in a meaningful way as part of the microgame mechanic (as in a rhythmic game)
(VII) use run time skeletal animations instead of sprite based animations in 2D

The reasoning behind that restriction is to encourage students towards building more complex microgames and, as a side effect, having more substantial diversity in their mechanics and art styles.

During this phase, the students had the freedom and responsibility to propose the microgames since its conception, going through its game design, implementation and testing, with the interference of the teacher happening only when their proposition either had a scope too big or too small for the available time (2 iterations).

Also, every microgame should have a configurable difficulty level ranging from 0 (very easy) to 1 (really hard), varying the level of challenge imposed to the player. Such requirement enables the progression of difficulty of the microgame sequence, which is typically randomly selected from a pool, as well as to allow difficulty adjustments by the player.

The game should target multi-operating systems of desktop environments, but also Android and iOS devices – hence, the microgames mechanics should be designed in such a way to be playable/enjoyable through mouse and also touch screens. The variety of target platforms also imposes an extra challenge of making the game graphics and input adaptable to different screen resolutions.

In the third and fourth iterations – phase II, the same groups had to pick different tasks related to polishing the game and turning it from a prototype quality level to a finalized game ready for distribution. Such tasks were defined in a meeting in which the whole classroom suggested and prioritized features that should or could be implemented to improve the quality of the whole game. Some examples of tasks were the composition of the main soundtrack, a better menu screen, a splash screen, sound effects between microgames, an online rank, a campaign mode, visual effects for screen transitions and several others. During this iteration, there were also tasks about polishing existing microgames as well as developing new ones, but the possibility of the latter was only unlocked when all of the higher priority tasks had been concluded. Lastly, in the same iteration, there were tasks regarding the game adaptation and distribution on the smartphone app stores.

As the tasks of the third and fourth iterations typically require changes in several files throughout the whole source code base, it is common to happen *merge conflicts* – when different people change the same lines of code in a file. At this point, the students have acquired some maturity with the VCSs concepts and commands and tackling the resolution of conflicts is an additional challenge and responsibility of this phase, as well as an opportunity for improving their experience in developing larger software and the daily activities of a professional software/game developer.

Even though the mechanics, restrictions and project schedule had been established, the game also needed a thematic axis to drive the students' creativity in proposing the microgames, as well as to improve the chances of captivating players. Hence, the Microgamr proposal strongly suggests such a thematics to be proposed, which can be done either by the teacher alone or together with the students. In the first two applications of Microgamr, the teacher himself defined the game themes, and his decision aspired to incite engagement in the students through empathy with the chosen subjects. Section IV presents both themes and details each of the two applications of Microgamr so far.

The following subsection presents some technicalities of Microgamr and describes with more depth how we conducted the project management.

C. Development and Project Management

Initially, the teacher created a seminal project with the source code containing the fundamental routines for the creation of a game with the microgames mechanics. The seminal code comprises a very raw, unpolished game, but with beginning, middle and end: a "splash screen" with a temporary logo, a "main menu screen" without any buttons – the player clicks or touches anywhere on the screen and is taken to the "playing a sequence of games-screen". There are two very simple microgames provided as samples: one with a winning condition called `WIN_WHEN_TIME_RUNS_OUT` (i.e., the user must "survive" until the timer ends) and another with the condition `LOSE_WHEN_TIME_RUNS_OUT` (i.e., the user must do something before the timer runs out).

We chose Java as the development language for its robust implementation of the object orientation paradigm, its popularity both in the industry and in the academy, the presence of disciplines that either enforce or suggest its use in the Computer Engineering graduation course and the availability of game frameworks targeting the Java platform.

From the game development tooling perspective, we like to think of them as an abstraction spectrum, in which in one end, we could use lower-level tools like OpenGL or DirectX and in the opposite end we could use very high-level tools such as YoYo Games' GameMaker [19]. We considered neither end of the spectrum interesting, as the former would require too much effort to put anything on the screen while the latter would significantly decrease the flexibility and control over the game being developed [1]. Above the graphics API level in the spectrum, there are "game frameworks", which are still low-level tools, but which provide higher productivity by decreasing the time and complexity of the lower-level programming tasks. Above it, but still below the "game makers" end of the spectrum, there are game engines. We chose to use a tool in the "framework" level, as it allows the coding to be more productive while not being as complex and hardware-demanding as an "engine" is.

In the Java ecosystem, we chose a framework called LibGDX [20] which is a popular choice among Java programmers for being well designed, open-source and multiplatform – it targets desktop environments (multiple operating systems), web browsers, Android and iOS devices. It provides one level of abstraction on top of the OpenGL ES 2.0 specification, as well as several useful routines typically necessary to develop games, such as vector and matrices operations, sprite batching, tilemap loading, physics engine, support for HUD building, pre-loading of assets, loading of multiple image, sound and 3D model file formats, among others.

As previously mentioned, we crafted the seminal code with an architecture that uses object-oriented programming concepts to allow more natural extension and modification of the source code. The remainder of this subsection presents such devices.

In terms of chronological schedule, each pair of iterations occurred contiguously (14 days) with a gap of 1 week between the next pair. Hereon, we describe the relevant technicalities for each iteration.

1) Iteration 1: creating microgames: To create a microgame, a student should create an extension of the `MicroGame` abstract class and implement the `MicroGameFactory` interface, following the hierarchy depicted in Fig. 4. In the `MicroGame` subclass, the student should implement its abstract methods and eventually call `super.challengeSolved()` when the user wins the challenge or `super.challengeFailed()` when the user fails it.

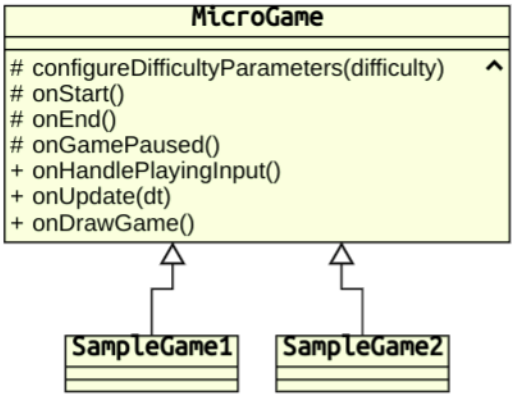

Fig. 4. Class diagram regarding microgame creation

```
public class MyAmazingMicroGame extends
    MicroGame {
    // ...

    @Override
    public void onStart() {
      // instantiate the game entities
      // like the player, enemies, background,
         sounds
    }

    @Override
    public void onHandlePlayingInput() {
       // handles the player input
    }

    @Override
    public void onUpdate(float dt) {
       // updates the microgame logic
    }

    @Override
    public void onDrawGame() {
       // draws the entities
    }

    // ... other methods
}
```

Additionally, every `MicroGame` instance must have a corresponding factory class (Fig. 5). That is necessary as the instantiation of a microgame is controlled by an entity called

GameSequencer, which needs to know how to instantiate each MicroGame. An elegant solution to allow code reuse and to isolate classes responsibilities was to implement the GoF *AbstractFactory* design pattern [21] for each MicroGame, which is in charge of knowing how to instantiate each microgame (Fig. 4). To do so, one must implement the interface MicroGameFactory in the following way:

```
public class MyAmazingMicroGameFactory
    implements MicroGameFactory {
   @Override
   public Microgame
       createMicrogame(BaseScreen screen,
     GameStateObserver observer, float
         difficulty) {
       // returns a new instance of
       // MyAmazingMicroGame, e.g.:
       // return new MyAmazingMicroGame(...);
   }

   @Override
   public Map<String, Class>
       getAssetsToPreload() {
       // returns a set of assets to be
          preloaded
       // before this microgame starts
   }
}
```

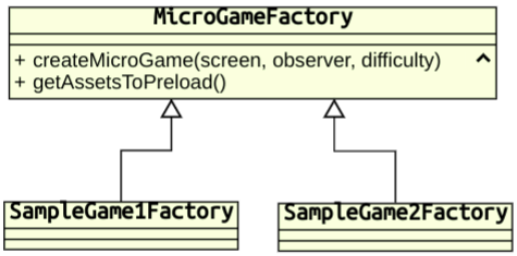

Fig. 5. Class diagram regarding microgame factory creation

Finally, when transitioning to the GameScreen, which is responsible for managing the sequence of microgames and the winning/losing conditions, students must state which MicroGameFactorys to use in its pool of available microgames. To achieve that, the programmer must add the newly created *factories* to the array of available ones in the proper method of the MenuScreen class:

```
// ...
public void startPlaying() {
   // ...
   GameSequencer sequencer = new
       GameSequencer(5, new
       HashSet<MicroGameFactory>(
         Arrays.asList(
            new SampleGame1Factory(),
            new SampleGame2Factory()),
            // write this new line
            new MyAmazingMicrogameFactory()
   ), 0, 1);
}
```

Then, students can implement the logic behind each microgame in its corresponding child class of MicroGame – using the onUpdate method to update positions/check winning/lose conditions, the onDraw method to render the game entities and so on.

2) Iteration 2: fixing/polishing microgames: By the end of the first iteration, students have received preliminary feedback of their microgames, and in the second iteration, they proceed on fixing the identified issues and polishing the games, according to the teacher's suggestions.

After the deadline of this iteration, the teacher grades the pairs of students considering what they have delivered and in case they did not receive a perfect score, they can work on the remaining items to increase it, but with a 25% penalty in the grading of such items.

Also, during the 1-week interlude between iteration 2 and 3, the whole classroom proposes new features to be implemented, and we generate a new backlog of activities to be developed in the next two iterations.

3) Iteration 3: general features and more polishing: A good portion of the proposed features involves the creation or improvement of game screens. The seminal code is organized in such a way that allows handling the logic of each screen in its class, which inherits from BaseScreen. All screens extend such abstract parent class, as it provides facilities for, e.g., navigating to another screen while unloading its assets and preloading the ones required by the next.

By extending BaseScreen, the child class must implement a few abstract methods:

```
public abstract class BaseScreen extends
    ScreenAdapter {
   // ...

   // called when the screen is about to be
      shown
   // for the first time
   public abstract void appear();

   // called only once and after all of the
   // required assets have been loaded
   protected abstract void assetsLoaded();

   // called only once when this screen
   // is disposed
   public abstract void cleanUp();

   // called every update frame to allow
   // the detection of player input
   public abstract void handleInput();

   // called every frame and responsible
   // for updating the screen state and logic
   public abstract void update(float dt);

   // called every time the screen must
   // be redrawn
   public abstract void draw();

}
```

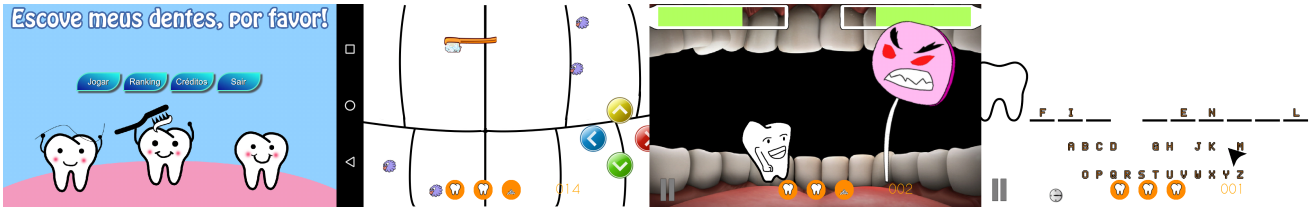

Fig. 6. Title screen of "Brush my Teeth plx" and some of the student developed microgames

Other tasks usually proposed for development include the composition of a musical theme, of short music clips to be played between microgames, and the creation of sound effects for when players either succeed or lose in a challenge and when he/she successfully finishes a whole sequence or loses all lives before that. Such tasks require skills that are not typically developed in the curriculum of a Computer-related graduation course. However, as the whole classroom is involved in developing the same game, there is a higher chance of one or more of the students having the necessary skills, experience or at least interest in picking up such tasks.

Other tasks revolve around graphics arts, as the creation of a team logo, a splash screen video, a menu screen which is more fluid and more labored. As with the case of the music/sound-related tasks, it has not been difficult to find students interested in the graphics-heavier duties.

Another group of tasks is mostly related to programming, such as the creation of a credits screen, a visual effect for screen transitions, a local and a networked rank, a smarter algorithm for picking up the next game in the sequence, the campaign mode, among others.

If the number of tasks is insufficient for the number of student pairs, there are also two "wildcard" tasks they can develop: polishing an existing microgame and creating a new one from scratch.

Each of those tasks has a different maximum score associated with it. The teacher assigns the score of each task using criteria that comprise the complexity and amount of time necessary to fulfill it. Therefore, when students choose which to pick up, they already know how much they might score if they complete the tasks accurately.

4) Iteration 4: platform-porting and bug-fixing: In the fourth week of work, most of the pairs have already achieved scores of 100% or more or are still wrapping up what they had started on the previous iteration. It is common for some tasks to have dependencies on others and, in such cases, some tasks might only become available on the fourth iteration – for example, the task of making a networked rank depends on the local rank to have been completed in the previous iteration.

Also left to this iteration are the tasks regarding porting and publishing the game to the smartphone operating systems and their corresponding app stores. Moreover, developers can fix bugs they identified in this iteration.

The next section describes the use of Microgamr in a Computer Engineering course given by one of the authors.

IV. Applications: Brush my teeth plz and Meow Au

Microgamr has been applied twice with students of a Computer Engineering graduation course in CEFET-MG, in a one semester-long elective class called "Digital Games Development". The class syllabus comprises topics on game design, game development tooling, artificial intelligence, computer graphics, networks, and physics.

Besides the work on Microgamr, students also had a final project and weekly programming assignments to be done in the computer lab. Such assignments aimed at exercising the concepts presented in the theory class and also to make students acquainted with the framework used in Microgamr (LibGDX).

The two applications of Microgamr happened in the second semesters of 2016 and 2017. Both classes had 20 students. Such course comprises 60 hours, with 30h in an expository classroom and 30h in a computer lab. Every week has a theory and a practical class.

In 2016 the Microgamr instantiation produced a game called **Brush my teeth plz** and in 2017 the game **Meow Au**. The next subsections describe both.

A. Brush my teeth plz

The first game created – Brush my teeth plz – targeted the audience of young children who get nervous around going to the dentist. A small excerpt from the statement is:

"A typical child is born pre-programmed to be scared of at least three things in the world: (a) the Boogie Monster, (b) Dentists and (of course) (c) Clowns. Some of those fears are detrimental to the child's formation, as is the case of the (b) Boogie Monster who, in reality, has never been seen in the material world, and the fear of (b) Dentists, which are there only to help. The other fear (c) stems from colorfully dangerous and terrible creatures, and the children do well in keeping it even as grownups. Even game development teachers keep their distance from such creatures. In this work, we want to help children enter very calmly in the dentist room. Then, we are going to create a simple, uncommitted, and roguish game, for children (~~and grownups~~) to play in the waiting room."

Students and the teacher developed a total of 21 microgames for the game during the four iterations, with the most part being created in the first two iterations. Fig. 6 shows the title screen and a few microgames.

During the polishing phase, students created a theme song, logo, general sound effects, developed new screens, improved the existing ones, an infinite mode, and an online ranking,

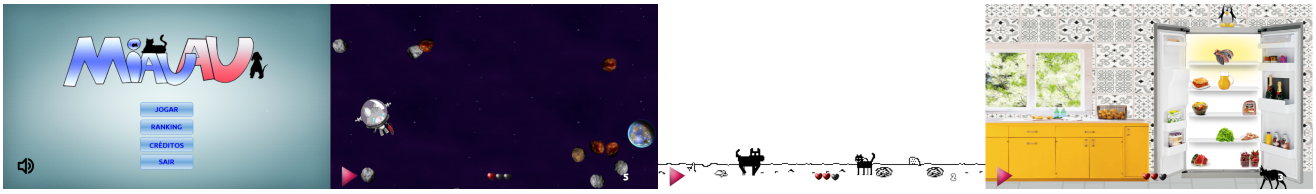

Fig. 7. Title screen of "Meow Au" and some of the student developed microgames

among other tasks. A pair of students generated an Android build and published the game at Google Play [22].

The project management was conducted virtually through GitHub issues[1] on the corresponding repository and students delivered code through pull requests.

During the iterations, the teacher acted as a technical leader to the students as he had more experience with the technologies involved and with game development in general. There were two meetings per week, which allowed the teacher to clarify whatever doubts students had very quickly.

After the development finished, the specification was improved to include the need for super-premium microgames, as we noticed some games were too similar and we wanted to instigate students to work on more advanced game development techniques. Also, we slightly improved the seminal code with learnings the team had while developing Brush my teeth plz.

B. Meow Au

Meow Au appealed to players who are fond of animals, especially pets. The work statement reads:

"Many centuries ago, humankind domesticated some animals, and they live together in harmony and tenderness. Dogs and cats are spread over the world and can be found in almost every home – but little is known about the secret life of such peculiar beings.

[...] *We always assume they sleep or stay quiet while we are not at home. However, that is not true. What are their goals – world domination? promoting human evolution? leveraging the scientific knowledge? preventing human extinction by destroying potentially dangerous asteroids?"*

In the microgame development phase, and later during polishing, students created 23 microgames, of which 13 were super-premium. Among the seven super-premium microgames categories, (I) physics engine, (IV) artificial intelligence, (V) 2 phases, and (VI) meaningful audio were implemented by at least one microgame. The others – (II) tilemap, (III) 3D, and (VII) skeletal animations – were all related to computer graphics and were not included. A likely explanation is that Microgamr was implemented before such concepts had been presented in the class, and students chose not to adventure themselves in (then) unknown territory. Fig. 7 shows the game title screen and a few microgames.

In the polishing phase, the tasks developed by the students involved creating a logo, icons, theme song, sound effects to be played between microgames and at their end, local and online rankings, a credits screen, an infinite and a campaign mode, an Android build, among others. It was also published [23] at Google Play, but this time by the teacher, as no group of students picked up such task.

After Meow Au's development, we evaluated the work from the perspective of students and the teacher, described next.

C. Evaluation of Microgamr

We evaluated the use of Microgamr in the classroom by assessing (a) the effort expended by the students and the teacher, (b) the technical knowledge in game development and collaborative programming acquired by students, (c) the experience of developing a single game for the whole class.

We extracted such information from data collected from the repositories and from a questionnaire in which students were invited to provide feedback on the work on Meow Au. The questions addressed the students' opinion on working on the game, and a total of 10 students participated.

In Brush my teeth plz, 250 commits were made to the repository in 53 pull requests. In turn, Meow Au generated 1,104 commits in 78 pull requests, which are numbers much higher than in Brush my teeth plz. Indeed, the interaction between students with the teacher and the code were a lot higher in the second application of Microgamr. Fig. 8 shows the lines of code inserted by the six most active contributors to the code, in which the teacher (user 1) had most of the insertions, as he was responsible for the seminal code and merging the students' pull requests. The hills in the area charts indicate the concentration of work during the two phases of the project for the students and in the interval between phases for the teacher.

The questionnaire asked students how they perceived the effort expended in the Digital Game Development class as compared with other classes they had. They considered it (50%) heavier or (40%) a lot heavier than other disciplines, but only partly because of Microgamr. Essential stress points they faced were learning LibGDX during the first phase and dealing with merge conflicts in the second. This is an interesting observation which validates the intended division of work between the two phases: students could focus on the game development technology at first, then on the intricacies of working collaboratively in a big project. A student also indicated the difficulty in creating or finding free assets to use.

Students' dedication in creating Meow Au was noticeable during its creation, but also in their answers and their commit behavior. Indeed, 80% of them wanted to have contributed more to the game. Reasons for not being able to participate

[1] A GitHub feature to describe a task to be developed or a bug to be fixed.

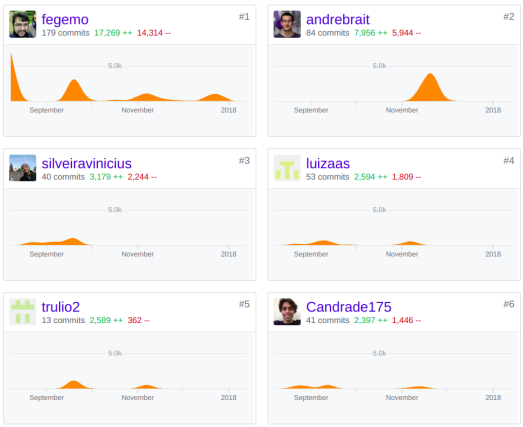

Fig. 8. Code insertions in "Meow Au" repository for the top six contributors

more in the game creation include mostly their involvement with tasks from other courses they were taking. Such engagement is also depicted in the punchcard of commits shown in Fig. 9, which reveals that some work was checked in the repository at almost all times of the day, even late at night and at dawns, and it was also spread over all weekdays.

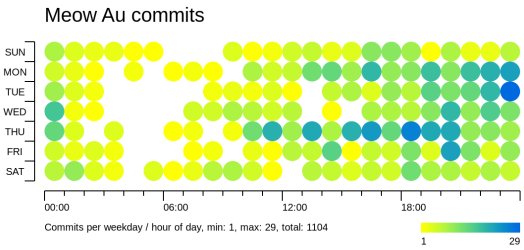

Fig. 9. Punchcard of commits in Meow Au

The experience of using Git to develop the game collaboratively deemed a nice evolution in the students' technical knowledge of the tool, as shown in Fig. 10. Some students already mastered it, but those who did not stated they learned how to use it. They also stated, as we predicted, that the first phase was easier than the second in terms of working collaboratively, which is evidenced by the excerpt from a student: *"[...] conflicts (in the first phase) were less frequent and easier to resolve when working in pairs. In the second phase, we needed to use and understand some git resources/commands which were still unknown."*

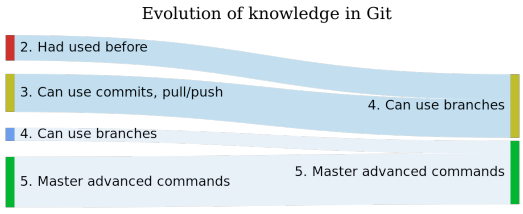

Fig. 10. Evolution of knowledge in Git of students

During the code reviewing activities from either phase, the teacher was able to evaluate the quality of the students' code. Those were great opportunities for suggesting good programming practices and the use of different object-oriented design patterns, which was not part of the course syllabus, but very useful and fortunate for programmers in professional environments nonetheless.

Lastly, regarding their experience of developing Meow Au, students mostly agreed that everyone creating the same game increased their sense of responsibility to deliver their tasks, as well as their motivation to work, as shown by Fig. 11.

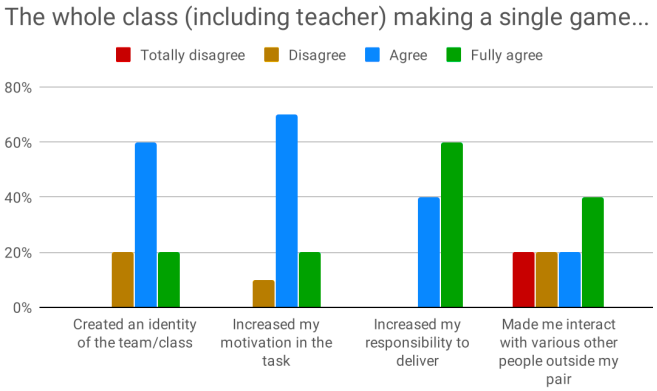

Fig. 11. Students' experience developing the same game

Resuming our research question of *"how can we effectively teach/learn intricacies of game development, including some tacit knowledge, closer to how a game studio works, in a classroom?"*, we can state that the experiences with Microgamr provided: a) deeper approaches for teaching concepts and sharing experiences among participants, as in the case of teaching content outside the syllabus (observed by the teacher); b) closer relation among students and teacher with more frequent and faster feedback (as observed by the teacher and stated in the questionnaire); c) high student dedication to develop the games (as seen by the number of commits and the punchcard); d) increased motivation and responsibility of delivering the proposed activities (as shown by the questionnnaire); and e) the student acquisition of skills of working in groups (questionnaire and observed by the teacher).

1) Limitations: As Microgamr requires the development of many different microgames (at least 15), it cannot be applied in classes with few students, as each one would have too much work. As a matter of fact, Microgamr could not be applied to the Digital Game Development class in later years, as there were not enough enrolled students.

Another limitation of Microgamr is that the concentration of roles played by the teacher – project management, tech leadership, programming, repository management, and a bit of art direction – can become cumbersome to the point of hindering its application in a classroom. A way to overcome that situation is to select a few students to play some of those roles. In the questionnaire, students reported being comfortable with taking such responsibilities.

V. Availability as a Learning Object

After the creation of Meow Au, we generalized the specification of Microgamr so it could be easily instantiated in other classes in the future. We created a hotsite[2] which hosts the code, some documentation, and general instructions regarding how to use Microgamr in the classroom as a learning object.

Teachers interested in using it as a learning object can follow the "getting started guide" which contemplates the steps of:

(1) creating a copy of the Microgamr repository;
(2) defining the game theme and name;
(3) splitting the class in groups and assigning them roles;
(4) executing the first two iterations;
(5) receiving all developed microgames;
(6) reviewing the microgames, providing feedback and evaluating students;
(7) proposing jointly with the class what tasks will compose phase 2 (polishing);
(8) executing the last two iterations;
(9) reviewing the implemented code, merging it to the main repository and evaluating the students.

VI. Final Remarks and Future Work

This paper proposes Microgamr, a learning object targeted at classes of students learning game development from a programmer perspective. The core concept of the proposal is to make a team from the whole classroom, including the teacher, to develop the same game collaboratively.

We have already applied Microgamr in two different classes of a Digital Game Development discipline from a Computer Engineering graduation course. Both experiences were successful as they engaged students to achieve a common goal, and the developed games were more prominent and more complex than if students were each developing their own game. Students' dedication was very high, especially in the second game, as the teacher was able to aid and demand more.

Also, the closer relationship with the students allowed the teacher to identify opportunities of teaching concepts outside the scope of the course, but still very important to their formation, such as good programming practices and the use of design patterns.

As future works, we intend on improving the Microgamr documentation by creating wiki pages in its repository, applying it again in future classes and, then, assigning different roles to students to decrease the work overload around the teacher. We also intend to propose a similar work for game development classes with a lower number of students.

VII. Acknowledgement

The authors would like to thank all students who worked on the games and enabled the creation of this learning object:

Brush my teeth plz: Amanda Pereira, Bruno Meneghin, Carlos Soares, Daniel Gonçalves, Gabriel Magalhães, Henrique Sampaio, Higor Amorim, Juan Ferreira, Lindley Vieira, Lucas Carvalhais, Lucas Campos, Lucas Viana, Luis de Carvalho, Matheus Rosa, Matheus Moreira, Nicolas Maduro, and Vinicius Pinto.

Meow Au: Adriel Augusto, Andre Brait, Arthur Abeilice, Cassiano de Brito, Emanoel Guimarães, Estevao Costa, Gabriel Melo, Gustavo Jordão, Gustavo Marques, Joao Matos, Luis Carlos, Luiza dos Anjos, Miguel Rodrigues, Natalia Natsumy, Pedro Belisario, Pedro de Jesus, Rafael Barbosa, Rogenes Reis, Sarah Rodrigues, Tulio Santos, and Vinicius Silveira.

[2]https://fegemo.github.io/microgamr

References

[1] J. Novak, *Game development essentials: an introduction*. Cengage Learning, 2011.
[2] D. Yu. Finishing a game. (2010). [Online]. Available: http://makegames.tumblr.com/post/1136623767/finishing-a-game
[3] ABRAGAMES. Relatório formação de profissionais em jogos digitais. (2013). [Online]. Available: https://drive.google.com/file/d/0ByQwQiUajB3mczhsRGpGVlVOTU03QW1VWUpweFYyWlp4ZVUw/
[4] S. de Freitas, "Are games effective learning tools? a review of educational games," *Journal of Educational Technology Society*, vol. 21, no. 2, pp. 74–84, 2018. [Online]. Available: http://www.jstor.org/stable/26388380
[5] I. Ouahbi, F. Kaddari, H. Darhmaoui, A. Elachqar, and S. Lahmine, "Learning basic programming concepts by creating games with scratch programming environment," *Procedia - Social and Behavioral Sciences*, vol. 191, pp. 1479 – 1482, 2015, the Proceedings of 6th World Conference on educational Sciences. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042815024842
[6] S. Combefis, G. Beresnevičius, and V. Dagienė, "Learning programming through games and contests: overview, characterisation and discussion," *Olympiads in Informatics*, vol. 10, no. 1, pp. 39–60, 2016.
[7] GEDIGames. 1º censo da indústria brasileira de jogos digitais. 2014. [Online]. Available: http://www.abragames.org/uploads/5/6/8/0/56805537/i_censo_da_industria_brasileira_de_jogos_digitais.pdf
[8] L. Torvalds, J. Hamano *et al.*, "Git," *Software Freedom Conservancy*, 2005.
[9] GitHub, Inc. GitHub: Social Coding. [2007]. [Online]. Available: https://github.com
[10] D. Spinellis, "Version control systems," *IEEE Software*, vol. 22, no. 5, pp. 108–109, Setembro 2005.
[11] S. Otte, "Version control systems," *Computer Systems and Telematics, Institute of Computer Science, Freie Universität, Berlin, Germany*, 2009.
[12] SBC, "Currículo de referência da sbc para cursos de graduação em bacharelado em ciência da computação e engenharia de computação," 2005.
[13] R. W. Gerard, "Shaping the mind: Computers in education," *N. A. Sciences, Applied science and technological progress*, pp. 207–228, 1967.
[14] D. A. Wiley *et al.*, *The instructional use of learning objects*. Agency for instructional technology Bloomington, IN, 2002, vol. 1.
[15] IEEE, "IEEE standard for learning object metadata," *IEEE Std 1484.12.1-2002*, pp. 1–40, Sept 2002.
[16] RhodeCode. Version control systems popularity in 2016. [2016]. [Online]. Available: https://rhodecode.com/insights/version-control-systems-2016
[17] Nintendo and Intelligent Systems, "WarioWare: Smooth Moves," 2006.
[18] J. Frost and S. Baird, "Dumb Ways to Die," 2013.
[19] M. Overmars *et al.*, "Game Maker."
[20] M. Zechner *et al.*, "LibGDX," 2013.
[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
[22] Coutinho, Flávio et al., "Brush my teeth plz," 2016. [Online]. Available: https://play.google.com/store/apps/details?id=br.cefetmg.games
[23] ——, "Meow au," 2017. [Online]. Available: https://play.google.com/store/apps/details?id=br.cefetmg.games.meowau20172