Simplifying location-based serious game authoring

Pérez-Colado Víctor Manuel
Dept. of Software Engineering and AI,
Facultad de Informática,
Universidad Complutense de Madrid
Madrid, Spain
victormp@ucm.es

Pérez-Colado Iván José
Dept. of Software Engineering and AI,
Facultad de Informática,
Universidad Complutense de Madrid
Madrid, Spain
ivanjper@ucm.es

Martínez-Ortiz Iván
Dept. of Software Engineering and AI,
Facultad de Informática,
Universidad Complutense de Madrid
Madrid, Spain
imartinez@ucm.es

Freire-Morán Manuel
Dept. of Software Engineering and AI,
Facultad de Informática,
Universidad Complutense de Madrid
Madrid, Spain
mfreire@ucm.es

Fernández-Manjón Baltasar
Dept. of Software Engineering and AI,
Facultad de Informática,
Universidad Complutense de Madrid
Madrid, Spain
balta@ucm.es

ABSTRACT

Pervasive gaming has been a field of exploration for the gaming industry the recent years. With the exceptional success of Pokémon GO and some others such as Ingress users are more than ever ready to experiment with games that are more focused into their actions and their playing contexts. In the serious game industry, the implementation of pervasive games is a very promising way to improve students learning process with a more cohesive and realistic experience. To take advantage of this opportunity we have developed a model for location-based serious games intended to be used along with other game genres such the adventure one. We have implemented this model in the uAdventure (uA) authoring tool, allowing non-experts for the simple creation for location-based adventure games without programming or complex geopositioning knowledge requirements. With uA we address to simplify the creation of gymkhanas, tours or guided visits with the added value of the adventure games. Finally, to improve the quality of the games, we have incorporated learning analytics into the system and tested the platform with a small experiment that showed promising results.

Keywords

Serious games; location-based; augmented map; GPS; augmented reality; ubiquity; pervasive; xAPI; learning analytics;

1 INTRODUCTION

Serious games market is expected to reach $5,448.82 Million by 2020 [19]. One of the crucial factors is the consolidation and constant grow of mobile technologies; accentuated by the usage of BYOD (bring your own device) in education and companies. Serious games applications are diverse; using serious games in schools is an innovative way to improve engagement of students; on the company side, serious games can be applied in the instructive and formative stage, being excellent to simulate corporate processes. For this reason, serious games may come in different formats and genres, focusing more into visuals, storytelling or in-game mechanics depending on the game purpose [5].

Mobile technologies allow for videogames and serious games to use novel mechanics and features; those involve using the context sources from the built-in sensors such as camera, radio sensors, GPS, accelerometer or compass; exploiting the players' ubiquity present in mobile. These capacities have impacted the growth of Augmented Reality (AR), now available for every smartphone user. Pervasive games defined as games that blends its in-game world with the physical world [12] is the super category of AR games and highlights the categories of location-based games, mixed reality games and affective games (games that involve user's physical responses).

Videogames in 2016, and especially pervasive games, have been highlighted by the global Pokémon GO[1] release by Niantic. The impact of Pokémon GO has been relevant in young people; it has had more than 650 million downloads and still has 65 million monthly active users by June 2017[2]. One of the most direct impacts has been detected in players heath [1]. Other case of success has been Ingress, the previous game of Niantic. Hence, Pokémon GO's (and Niantic games in general) success gives the community the chance to start developing other kind of games in the Pokémon GO format having more chances to be socially accepted [10].

Pokémon GO is well-known as a AR game; however, it fits even better in the location-based games as its AR factor has been revealed not as its crucial mechanic (many player have deactivated it) being the augmented map (AM) the most important feature. The key difference between AR and AM is that, both focus into overlaying in-game elements in real-world, but the AR focuses in live images from the phone camera, while AM focuses on map information (that comes also from reality). While the AM is a typical feature of the location-based games, AR can be fitted into several categories (location-based, mixed reality, etc.) depending on the end use of the AR inside of the game. For instance, if the game mixes a specific real world location like a park or a museum with AR, it can fit in the location-based, but, if the game uses AR as game environment (like in Playstation's Invizimals[3]) it fits better in the mixed-reality.

Location-based games AM is the in-game main view or scenario based on a map that includes virtual elements on it. The map itself may come in different forms depending on the source: physical

[1] http://www.pokemongo.com

[2] http://expandedramblings.com/index.php/pokemon-go-statistics/

[3] https://www.playstation.com/en-gb/games/invizimals-psp/

maps, based on satellite images and can be increased with height information to create more realistic maps (i.e. Mapbox[4] SDK); schematic maps, based on an abstraction of the real world information, creating more simple figures and shapes, and often including real world information (such as names, places or POIs); vector maps (in the middle between physical and schematic) include information about landforms, paths, buildings, parks, etc., stored as vector instead of as an image. In addition to the map, there are the in-game elements generally placed in the map, and may be linked to a point or zone.

The AM normally shows an avatar in the center of the map to represent the player; located using satellite. GPS (Global Positioning System), GLONASS and Galileo are positioning systems is based on radio signals emitted by satellites. They can achieve a precision of a few centimeters but it commonly works with a few meters accuracy, decreasing when the signals are not in line of sight (i.e. indoors); and it can provide measures of movement speed, direction and altitude. However, other approaches are possible without using satellites. Close range radio devices such as Wi-Fi's and Beacons can be used to locate the player too [2, 23], and the augmented map might be just a support tool for orientation or might not be present at all. The benefits of this kind of positioning is that works the same indoors and outdoors. In addition, it may be implemented by using QR codes that are cheap and accessible for independent developers.

Developing location-based videogames can be achieved in two ways: (i) creating tailored games that fit exactly the needs using SDK's (i.e. Mapbox, Google Maps, etc.) or game engines (Unity, Unreal, etc.) but require high level knowledge, time, and invest or (ii), using an specific platform for this purpose, intended to simplify the process by providing templates of all the possible tasks this games involve.

There are several platforms in the market available such as DiscoveryAgents[5], ActionBound[6] or GooseChase among others. With them, developers can easily create location-based experiences (e.g. gymkhanas, tours, treasure hunts, visits) by including mechanics that require the player to interact with the environment (e.g. reach locations, take pictures or videos, scan QR codes). In general, the experience includes several customizable tasks (normally sequential) the player must perform during the play. Despite the differences elements they may or not include, most of the platforms include analytics that allow the game developer to track players, inspect their responses and see leaderboards.

All these platforms fit in the serious game market, but there are some specifically developed for teaching, such as DiscoveryAgents, MIT's MITAR and eAdventure for Android. DiscoveryAgents aims to teach players about public parks and natural reserves. An experiment in the Calgary park evaluated with tests reported more fun and better emotions than the players that didn't use the app; and in terms of knowledge they reported similar results to the students that visited the park with the park guide [6]. MITAR platform was developed in 2008 [8, 13] by using AM with PDAs (Personal Digital Assistants) and highly involving narrative content. MITAR has been applied in the learning process in two ways: (i) using its output games; and (ii) using the platform for learners to create games themselves. In the first players had to interact with the playground through their PDAs to acquire information about the environment, talking to virtual characters our using simulated sensors; this helped to achieve a more connected and authentic learning, highlighting players' engagement. In the second approach students had to choose a topic and create a game that their classmates were going to play in the end of the experiment; conclusions shown that students tend to get deeper knowledge to explain it in the game, and they need to experiment themselves the playground to replicate that in the game.

The previous examples revealed that using location-based serious games helps users to better settle down the subject by complementing deducing and experimenting in the real world with the in-game content [4]. In addition, AM also provide an interactive way to teach learners about cartography, orientation and investigation.

As seen in the previous examples, we expect that including more narrative content in location-based content will result in richer and more interesting serious games. eAdventure (eA)[7], developed as Open Source in Java, was launched in 2007 and allowed the creation of serious 2D point-and-click adventure games eA [21]. A test version was developed for Android intended to use location-based features such as scene loading depending on player's location and QR code but, due to Java issues and project discrepancies, the branch was discarded and the project has been reimplemented in Unity as uAdventure (uA) [16], offering new possibilities for multiplatform. For this purpose, we propose using uA platform as the perfect target to introduce location-based mechanics into it, such as AM. The resulting games will not only fit in the location-based category, but also in the adventure category, resulting in a new game genre we call the location-based-adventure or located-adventure.

In this paper, we propose a generic model for located-adventures and location-based games, exposing all the AM elements, interactions and different positioning methods and provide an example implementation. In addition, editors for these features will simplify the task of creation this format of games. Finally, an analytics model will be proposed for all the new interactions to be merged with the already existing analytics xAPI model present in uA.

2 A GENERIC LOCATION-BASED MODEL FOR UADVENTURE

This section exposes the location-based game model as a generic approach to be used by game engines and editors and, in this case, by uA. As location-based games may be approached through different ways (i.e. Wi-Fi or Beacons), this paper will expose models for GPS and QR code location-based.

2.1 Positioning outdoors: GPS based model

Location-based key element is the AM and indeed, is the most important feature while playing outdoors as it both guides the player and provides information.

This AM can be included from an external library (i.e. Google Maps, OpenStreetMaps[8], Mapbox, etc.) or be self-implemented. Due to maps contain big amounts of data, normally this

[4] https://www.mapbox.com/

[5] https://www.discoveryagents.net/

[6] https://en.actionbound.com/

[7] http://e-adventure.e-ucm.es/

[8] https://www.openstreetmap.org/

information is segmented into tiles (regions) of different sizes (levels of detail) based on the zoom property. This segmentation is specified in the TMS (tile map service) specification [7, 20] and transforms tile coordinates to world coordinates; and so the world coordinates are expressed in meters that can be used in most a game engines. For self-implementation cases, we recommend to translate and scale world coordinates to local coordinates to avoid the overflow of the meters, by translating the map center to the world origin and then, use the scaled meters from the center to locate the elements over the map using coordinates. Once the map is ready, to augment it, it is necessary to include game elements. As maps mainly based on coordinates, the most useful classification for AM elements is whether the element is fully based on coordinates or is just an external element with location-based metadata.

Based on this classification, we propose a hybrid model using map-native elements fully based on coordinates (from standards GML and GeoJSON) and game-native external elements extended with location-based features. By combining both types it is possible to create gymkhanas, city tours, treasure hunts, etc. with rich visuals, ensuring engagement and a better cohesive learning. For instance, a city tour has a set of coordinates -map-native elements- connected via navigation with information attached, and a set of graphical elements -game-native elements- in the map representing features.

2.1.1 Map-native elements. Map-native elements are designed to be used inside of a map and are based on coordinates. These elements, from now on called GeoElements, rely on a set of latitude and longitude tuples to be represented in the map and are based in different standard definitions such as Geo-JSON or GML that also uses a set of coordinates to define map elements. Since these standards are very complex to implement due to the wide range of geometries [3, 14], we determine three essential geometries that are enough to implement several use cases for serious games: POIs (Points Of Interest), Zones and Paths:

1. The POI is a single coordinate element in the map of interest or where the player has to perform an action. Since POIs are just a single coordinate, an influence radius is added for more flexibility about the POIs' range of influence.
2. The Zone is a range of ordered coordinates that determine an area. In-game, a zone is a special ambit where the player could be contextualized; being in or crossing the border of a specific area may be the trigger for an in-game event. Essentially, a zone is a polygon and the GML standard suggests defining them by a single exterior polygon and a set of interior polygons to be subtracted from the original polygon; in turn, being defined in two possible ways, by a linear or shapely ring. For simplicity, we consider the exterior polygon to be enough in most cases. However, a simpler way to determine a polygon roundness is to establish a generic influence radius, being unified with POI's definition.
3. The Path is a range of coordinates, like a Zone except that paths are not filled. Because of this, the path has a starting point and an ending point, determined by its order. In-game, Paths can determine barriers the player should not cross; routes the player must follow; or just informative routes for the player to know (e.g. suggested gravel paths, bike paths or rivers to avoid). The standard GML offers two ways to define Paths: linearly, as LineString, or shapely, as Curves. Due to curves editing complexity, we consider LineStrings are enough. As in the previous cases, Paths also should have an influence area to determine if the player is inside the path or has left it.

All three elements, then, have a geometry and an influence area. Based on these features, we have determined four different actions the player can perform with these elements: enter, exit, look-to and inspect:

1. The enter action is performed when the player crosses the limit of the influence area towards the inside of the geometry, coming from the outside (as the player could be inside at the start of the game).
2. The exit action is performed as the player leaves the influence area (where the player was before).
3. The look-to is an action that involves the use of a compass to determine if the player is facing an specific direction. This action can be performed with elements (i.e. if the player must face a game element such as the center of a polygon or path) or directions (i.e. if the player must look from a location to a desired direction such as a building's side). For the latest, a vector specifying the direction is used. This action can be also configured to be limited to the element influence area.
4. The inspect action is an action to be performed interacting with the element on-screen. This way, a click performed to the element can be used to trigger an event (for instance, clicking into a POI can display information or pictures).

This set of actions allows the connection of geographic elements with an interactive game experience fitting location-based game's needs. Nevertheless, the game native elements are also crucial for a better game experience. In the next subsection, we explore their characteristics and potential actions.

2.1.2 Game-native elements. Game-native elements are objects that already exist in the game engine meant to be used in virtual worlds, without any constraint for the gaming context; they include positioning configuration, physical information (for the physics engine) and behaviors to define aspects for each object. For instance, Unity uses GameObjects[9] with Components to represent any game element. Hence, since the augmented map uses its own positioning system based on coordinates, these elements need to be transformed to be used adapting the virtual world position to this coordinates-based position.

The transformations of game-native elements mainly involve translating, scaling and rotating towards their desired position into the AM. Depending on the in-game purpose of the object, these transformations will vary with the possible user interactions. For example, in this AM environment, the elements are fixed in the map, by default, but also there could be other elements such as screen buttons, arrows or indications depending on the user position.

Therefore, we define three different positioning systems: map positioned, based purely in coordinates; radial positioned, based on other element position and other physical measures; and screen positioned, that does not depend on the map:

1. The map based positioning adapts the world coordinate to the AM local position including a scale factor (that works accordingly to the map zoom) and a rotation. To

[9] https://docs.unity3d.com/Manual/class-GameObject.html

control the interaction, by using the player's distance to the object we can allow or disallow it. In addition, this distance limit can be used to hide and show the element whenever the player is in-range or not, such as Pokémon GO does. We identify this maximum interaction distance as the object influence, being represented radially.

2. The radial based positioning adapts the position of an element nested to another element in the map (e.g. the player avatar). For this, we use three parameters for: distance, angle and an additional "rotate around" parameter (to rotate the element as the player rotates and make the elements completely relative to the parent object). In this case, it is not necessary to block the interaction depending on the influence area as objects are mostly visual or utilities (navigation arrows, pointers, particles, etc.).
3. The screen based positioning adapts the world position to the viewport position. There are two ways to approach this: (i) by reverting the map transformation process, for which it is only necessary to reverse the process described by the TMS, and (ii) by totally avoiding nesting the object inside of the map, and with that, avoid the parent transformations, which for the mainstream game engines, just requires the object to be set independent, and move with the camera.

The interaction with these elements is made through bypassing the user interaction to them from the map, resulting in these objects acting as they do naturally. For instance, in adventures the elements will provide their look, use, grab, etc. native actions.

Since the nature or behavior of the native object to adapt for the map is a priori unknown, we suggest to create an object wrapper that converts the original object coordinates to each of these map coordinates, avoiding to adapt each object individually.

2.1.3 Nested scene launching. Mixing other serious game genres mechanics with the location-based scheme could improve the learning process by taking the best of each genre. Using game-native elements in the AM is the minimum approach; however, a bigger visual change might be useful for cases where there is a complex task to do in a certain location that may require using a better visual representation of the concepts (e.g. a mini-game, adventure-like or AR scenes).

To do this we propose do nested scene launching; that may happen because of any interaction (location-based or not). As the scene is launched, the game changes its appearance leaving the AM and the player interaction; even so, location-based mechanics may be present in a standard-game scene. For this, we propose that launching a nested scene could be linked to a specific map-native element that retains the scene while the player is in the element and, when the player leaves, the scene switches back to the AM scene.

2.1.4 Navigation. Most native maps include a navigation system for the user to know the best route to reach the destination. This navigation system may come handy for games which purpose may not require the player to know how to navigate or find a location. Since the game author might not want to take care of each navigation point we suggest using game elements as route points.

To launch the navigation system, it requires set of elements to travel and one traveling order. We suggest two different orders: linearly, one by one; or by distance, the closest first (potentially reducing the game playing time). This order is clearly useful for gymkhanas where challenges do not need to be accomplished in a specific order. Also, since each step might involve a task to complete, each element needs indicate if the element is completed as the player reaches it or if it requires a manual completion managed by in-game mechanics. This requirement is especially notable in gymkhanas or treasure hunts, where the player must resolve a clue or complete a challenge to step ahead to the next point.

The navigation system, then, must implement a public interface allowing other game components to configure and control the navigation present in the game. To select or step the current destination, its interface should provide public methods, so each game-element determines if the task has been completed.

There are several ways to approach the navigation system, depending on the game purpose. For instance, games in parks have freeway so the navigation only needs a single arrow to determine the direction; however, games in cities may use a street based navigation with distances, turns, etc.

2.2 Positioning indoors: QR code based model

Since location-based games are ubiquitous, they might happen indoors too. However, satellite oriented positioning systems are not so effective in this context [12, 17, 18]. There are several alternatives to achieve a positioning system indoors such as IMU based, optical-based or radio-based. Field tests with QR codes are sustained on the arguments that is an easy, simple and efficient way to determine the players location [11, 12]. In fact, the QR code positioning system is widely used in location-based games as it could be used outdoors or even in combination with GPS to create a better user experience (i.e. hidden clues in treasure hunts).

This model approaches location by assuming that scanning a QR code requires the player to be physically located in the same position as the QR. Since scanning the QR code is part of the game sequence, either (i) the game has to be programmed to expect a QR code at a certain point and continue the sequence after it; or (ii) the QR is able to include the in-game consequences inside the code for the game sequence to advance (i.e. mechanics such as dialogs, image show, or just game state changes). For linear games, the first approach might be easier to design and implement, but, for location-based games, where the game sequence might involve freely affordable challenges, the second approach is better as it allows to implement this nonlinear game sequences as well as the linear sequences that may happen inside of each challenge.

In order to protect the game content associated to the QR code, as well as the linearity required in some cases, we suggest to use identifiers as QR content. This way, the game acts as a mediator to access the content, and can even protect the linearity by only allowing the content access if certain conditions are satisfied (i.e. if the player has passed all the previous tasks inside the challenge).

The other key piece of a QR code model is the QR scanner; there are two possible implementations for it: (i) using an external app and launch the game passing the content to the game; or (ii) using a embed scanner inside of the game.

Using an external scanner app is the easiest approach but it breaks the gameplay as the player has to exit the game to be able to scan the QR. To implement it, it is necessary to define a URI based namespace that tells the OS to launch the proper application and embed it into the QR code content. For instance: an uA URI for QR could be "uA://qr/random-id"). This protocol must be registered into each OS the game is targeted for, ending up making the process not so easy in the end.

Using an internal scanner (by using a library, for example) requires more effort to implement with derived costs of money and time. Nevertheless, users will not have to exit the game to use it, avoiding to reload the game or restarting the services (GPS, accelerometer, etc.). Furthermore, the scanner can be limited to the moments that is necessary, simplifying the gameplay; and can be restricted to scan the QR codes the situation involve.

2.3 Implementing location-based adventures in uAdventure

The described location-based model has been implemented in uA extending its model. A runtime infrastructure that is able to parse the stored model in xml and creates all the requirements for the location-based games has also been implemented.

uA's key elements for authoring are the condition system, that allows to control the game behavior based on the global game state; and the effect system, that allows to visual program the consequences of the user interaction.

2.3.1 Outdoors positioning. The uA game platform has been extended by adding a new scene named MapScene to store all the configuration for the AM and all the elements, being those GeoElements or native elements (NPCs, Items or Atrezzos). Both MapScenes and GeoElements can be configured by using the Map module implemented for the Unity Editor that can represent polygons, areas, points and images, and handle the clicks for creating points or selecting, and the drags to move points or elements.

As uA is a Point and click adventures game engine [16], by extending it with AM it will generate location-based adventures. Hence, the process of creating location-based games based on AM in uA first start with the creation of the native elements (and its possible interactions) that will be added to the MapScene. These elements then, can be added to the MapScene in the MapScene tab (Figure 1); and configured with a specific positioning method (world, radial or screen), that determines how the element is placed, and some of its interaction constraints (such as the influence area).

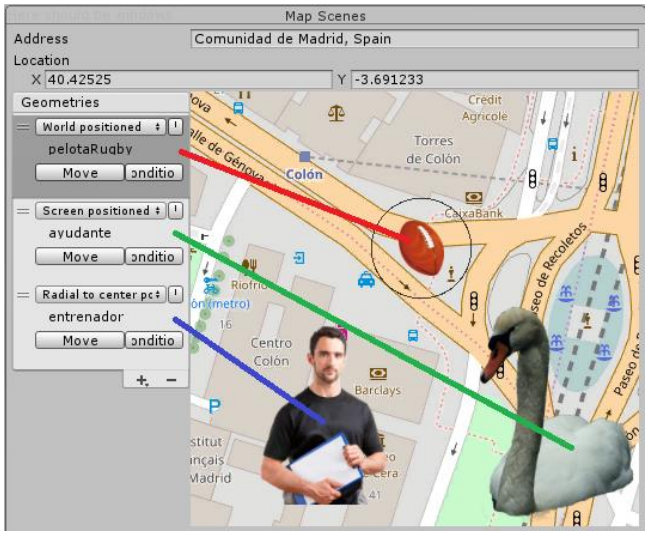

Figure 1: MapScene editor tab, displaying three native elements with all the possible positioning systems.

Secondly, the zones, points and routes, along with their location-based actions (enter, exit, etc.), must be created before added to the AM. To create them, the GeoElement tab (Figure 3) allows creating the different polygons and configuring their influence area.

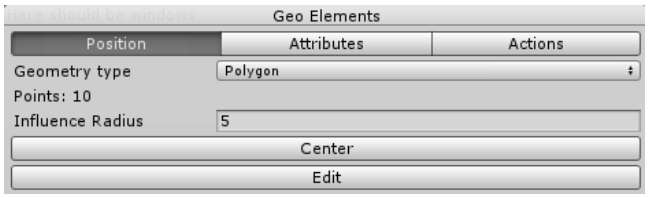

Figure 2: GeoElements main tab. An editor map as the one present in Figure 1 and 4 is present below the "edit" button.

For each GeoElement selected this tab geo-actions section (Figure 3) manages the different location-based actions associated to it. Each action will have a set of conditions to determine its access and a set of effects to determine the consequences.

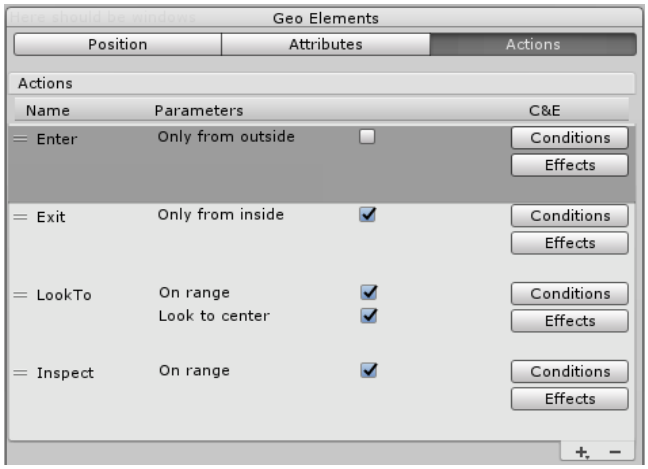

Figure 3: GeoElement editor tab. Top shows the geometry attributes. Bottom shows a set up of all possible location-based actions.

Once all the elements for a specific MapScene are prepared, they can be added into it by selecting them in the MapScene tab (Figure 4). The map scene view shows the editor map, where the elements can be configured. Both map and native elements are stored as a reference with a set of conditions that can be used to show or hide the elements depending on the game state. For instance: there could be a character used to teach the player how to play, but after the tutorial is finished, the character can be hidden by activating a flag after finishing the tutorial. Native elements will also store their positioning information in the reference.

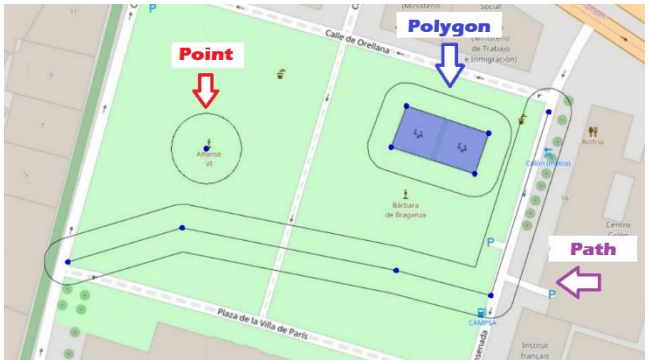

Figure 4: Location-based geometries. The red is POI, the blue is polygon and the purple is a path.

At the runtime stage, once a MapScene is loaded we instance a map based in tiles (Figure 5) based in the open source AM for Unity project of Baran Kahyaoğlu[10]. This AM has the potential to work with tile maps based in both graphical and vector information and can be customized depending on each game requirements. However, this map has been modified to include factories for uA elements (both native and location-based) that are instanced in the corresponding tiles using the TMS transformation.

In the map view, the map-elements are displayed by using polygons of different colors with a tooltip with the name above it: POIs are represented with spheres; Paths are horizontally extended and rounded; and polygons are just transformed into meshes using triangulation.

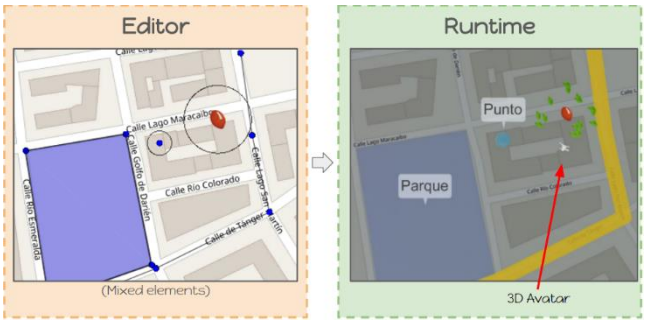

Figure 5: Correspondence of a set of elements in the editor view with the runtime view. Native element on top of the avatar is displaying the discover leaf-based effect.

On the other hand, the uA native-elements are wrapped runtime and positioned depending of the selected positioning system. It is especially remarkable that the implementation of the map-based positioning includes the hide and show mechanic based on the player closeness. A special effect using particles with the shape of leaves is displayed once the object is shown, similarly to the Pokémon GO discovery effect.

Scene nesting discussed in the model has been also implemented for map-elements. At the editing stage, a special variant of the trigger scene effect named trigger geolocated scene effect has to be configured by setting the desired GeoElement the scene will be nested to. Once the player leaves the scene runtime, the nested scene is closed and the player is switched back to the previous AM.

Navigation system also described in the model has been implemented inside of the effect system (Figure 6). The navigate effect is used to configure the navigation controller with the desired steps and the order of processing (being by list order or by closeness order). Secondly, for those steps that require specific conditions to be completed, the navigation control effect makes the navigator controller change its current target. At runtime, the navigator has been implemented simply by using a straight arrow that indicates the direction of the destination (Figure 6). Future versions will probably implement the usage of a route system for cities.

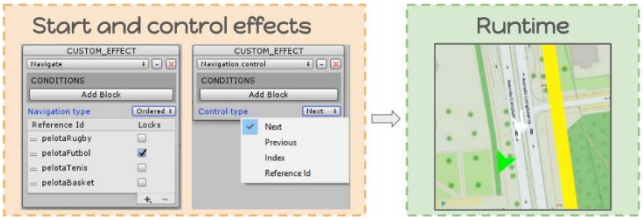

Figure 6: Navigation. In the left, the effect to set up and control the navigation. On the right, the runtime visualization of the navigation arrow.

2.3.2 Indoors positioning. As the model described, positioning based on GPS provides a good positioning quality outdoors but is not such convenient for using indoors. uA approaches to solve this by using QRs as described in the model.

The QR tab (Figure 7) inside the uA editor manages the creation and content of each code. Once selected, the tab allows to define the QR content formed by: a line of text that is shown after the code is scanned; the list of effects that will determine the consequences of scanning it; and the set of conditions that restricts the QR scanning that could break the game linearity. This content is explained hereunder.

The sequence of actions that happens after a QR is scanned determines how the game progress after the player scans it. For this, the model showed two approaches for linear and nonlinear situations. In this case, we considered that to afford nonlinear solutions (when needed), each QR contains the in-game effects (provided by the uA effect system) of scanning the code that controls the game progress.

On the other hand, to guarantee that the linear parts inside of the game are locked until the player reaches the point we used the uA condition system. This way, each QR contains the restrictions to be scanned. With this, the QR is ready to be used in the game.

Finally, the QR tab also allows for saving and printing the QR codes from the uA editor, simplifying the QR exporting and reducing the game setup times.

Figure 7: QR code editor tab.

As the model showed, the other key piece to work with QRs is the scanner. As uA is a gaming authoring tool, and we want to provide the best in-game experience and the easiest setup we decided that using an integrated QR scanner was the best choice. This integrated scanner can be opened in-game by launching a QR prompt effect (Figure 8), that contains a message to show to the

[10] https://github.com/brnkhy/MapzenGo

player, and the desired limitations for that scanning session. Since it is an effect, it can be linked to an on-screen button, conversation, or whatever the game designer requires.

In-game, the QR scanner shows the message in the top and opens any of the available cameras (preferring the back camera) showing an animated screen reminding a barcode scanner.

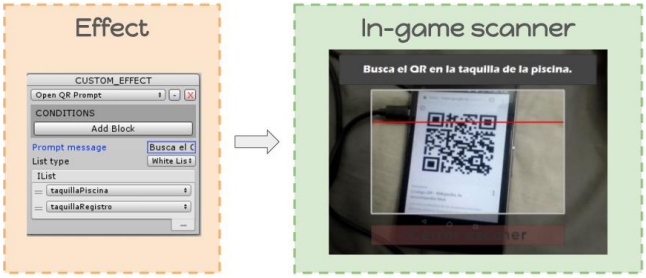

Figure 8: QR scanner. In the left, scanner prompt effect configured with a white list of possible codes. In the right, the in-game view of the scanner with the message on top.

3 LEARNING ANALYTICS IN LOCATION-BASED GAMES

Fitting a serious game in the learning process usually requires an assessment or evaluation process to verify the game has been effective. This can be approached different ways but new trends afford learning analytics (LA) as the next generation move since they can both cover traditional assessment, and be applied to more complex analysis techniques (i.e. datamining) that allows authors and teachers for a better understanding of the learning process.

There are several possible approaches for its implementation, highlighting xAPI[11] over all [9, 22]. For clarification, xAPI is a standard for traces that represent what actions or events are happening real-time, in a high-level language, identifying actors, verbs and outcomes. Hence, traces are easy to read and have direct value for LA.

uA implements the xAPI for serious games profile [15] from which are implemented four categories of events: (i) Completable, intended for tracing game progress, tracking each task in the game based on a group of milestones or steps; (ii) Accessible, for tracing the player in-game movement, such as stage changes; (iii) GameObject, for tracing the user's actions with in game elements; and (iv) Alternative, for tracing the player decisions, that are especially valuable for assessment.

In the location-based games ambit, however, the xAPI for Serious Games profile is insufficient as the verbs, terms and extensions are oriented to track in-game elements, missing aspects from the real-world environment actions (movement, looking, access, etc.). Hence hereunder we present the draft of what is going to be a new xAPI profile or location-based applications.

In location-based games we identify two different ambits for actions: (i) player's Position related, that include real-world coordinates information and contextualization about the area where the player is; and (ii) player's Direction related, that include the different choices the player makes while traveling. An example of the Position ambit could be a trace where the player has Moved to an Urban-Area with the ID Madrid, and with exact coordinates as a Location extension. On the other hand, for the Direction ambit, when the navigation system of the game decides a route that links the starting point with the next point-of-interest, traces could identify whether the player has follow those directions, or has chosen his own path.

Once the system has been integrated with this model the traces must be collected and analyzed. For this purpose, uA uses the RAGE Analytics system designed to work with xAPI traces. In this system, a location-based traces analysis is done to finally generate a set of map-based visualizations, especially useful to rapidly understand how players perform location-based actions.

A more exhaustive analysis of the visualizations can help the teacher to determine, for example, (i) using a heatmap, which of the designed missions are misleading the students to an undesired location, or (ii) using a route-map, if a student has gone straight to the point-of-interest, or he has got confused for a period, and reached the destination with a delay.

4 FIELD TRIAL EXPERIMENT

With the purpose of validating the new uA upgrade for location-based games, we developed a test game that uses most of the features implemented. The experiment purpose is to explain the sport facilities and the usage of the registry in the Moncloa UCM campus having four parts: (i) tutorial for using compass and a navigation to one of the multiple sport facilities; (ii) game of gathering hidden balls in the different fields; (iii) explanation of the swimming pool facility (inside) with QR scan; and (iv) simulation of a document delivery at a registry with QR scan.

The methodology of the experiment was performed in a three-step process: (i) pre-test of knowledge; (ii) game play and analytics collection; and (iii) post-test of knowledge and satisfaction. The development of the game, materials and setup was made in less than a week, including testing.

The trial involved four people and they decided the game together as a single team; in a session that lasted for 51 minutes. As a summary of the results, we detected an improvement on the response rate from a 29% to an 86% of correctness.

In the analytics report we could explore the players' movement, actions and behaviors in the location-based heatmap analytics (Figure 9); and time measures of each task completion, whose results concluded a bad ratio of navigation/challenge time that players also reported in the satisfaction test.

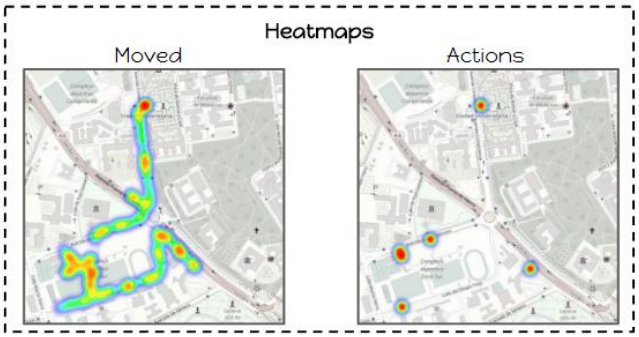

Figure 9: Heatmaps analytics reports. Left shows players route during gameplay. Right shows points where location-based actions were performed.

From the satisfaction test the experience got an overall 7.2/10 score. The players' comments in the positive side highlighted the ball collection part, as they enjoyed having to deduce where the balls could be. On the negative side, they highlighted the long

[11] https://www.adlnet.gov/xAPI

navigation periods without anything clear to do, that could be filled with some task about gathering random elements; and absence of a competitive leaderboard.

5 DISCUSSION

Location-based gaming market is growing enormously, specially highlighting Pokémon GO success, creating opportunities to the serious game market to move along and benefit from it. However, as this paper has discussed, there are no platforms oriented to create location-based games adequate for the low-cost serious games. In addition, these platforms also miss the opportunities arising from mixing up aspects from different game genres to better focus the objective of each learning situation (i.e. navigation vs simulation).

In this paper, we present a location-based model designed to be integrated in other serious game authoring tools supporting other game genres; or it also can be used as a model for location-based serious games themselves. This model, that is based on well-known standards aims to be compatible with existing map models, but also creates a layer of interaction especially valuable in location-based serious games (e.g. enter, exit, look-at). Secondly but not less important, this model aims to mix game genres and therefore it allows for the integration of native elements into the map and even nested launch other genre's scenes. Because of this, we've exposed different positioning techniques and behaviors mixed elements should use. Complementing the map representation, navigation systems helps the player to move in the maps and teaches players about orientation and navigation.

This model aims to work both outdoors and indoors, so we showed different approaches highlighting a possible implementation based on QR codes. By the usage of QR codes it is possible to complement the system and provide better game experience. Future work for the uA platform will involve the implementation or location based in Beacons or Wi-Fis too.

Narrative serious game is the genre we attempt to integrate location-based model into. Our authoring tool, uA has been extended to include several tools for authors to easily create map-based elements and mechanics and to integrate the adventure game elements into the location-based model. In fact, the editing tools provide nice visual representations of the AM design and tools such as reverse geocoding services to avoid dealing straight with coordinates.

Finally, to improve the serious game life-cycle, assessment is made by using the novel approach of integrating learning analytics. We've drafted a location-based xAPI profile that can be used for any kind of location-based applications and we've created the proper visualizations for non-experts to understand and evaluate players.

As a conclusion, with our small field test involving real users in a simple game context, we've obtained promising learning results (in terms of knowledge and engagement) and nice receptions from the player's side. Soon, we expect to test this model and provide more realistic conclusions comparing different learning methodologies. However, at this moment, the uA platform is still in early stage, needing a huge push in terms of robustness and usability, but still we hope this paper will impulse better learning experiences for students, more connected with the real-world and, in the end, more authentic.

6 ACKNOWLEDGMENTS

We want to specially acknowledge Baran Kahyaoğlu for his open source MapzenGO project. This research has been partially financed by the Regional Government of Madrid [eMadrid S2013/ICE-2715], by the Ministry of Education [TIN2013-46149-C2-1-R] and by the European Commission [RAGE H2020-ICT-2014-1-644187, BEACONING H2020ICT-2015-687676].

7 REFERENCES

[1] Althoff, T. et al. 2016. Influence of Pokémon Go on Physical Activity: Study and Implications. *Journal of Medical Internet Research*. 18, 12 (Dec. 2016), e315. DOI:https://doi.org/10.2196/jmir.6759.

[2] Brassil, J. 2014. Improving Indoor Positioning Accuracy with Dense, Cooperating Beacons. *Procedia Computer Science*. 40, C (2014), 1–8. DOI:https://doi.org/10.1016/j.procs.2014.10.025.

[3] Butler, H. et al. 2016. *The GeoJSON Format*.

[4] Coulter, B. and Klopfer, E. 2016. Discovering Familiar Places: Learning through Mobile Place-Based Games. *Games, Learning, and Society: Learning and Leading in the Digital Age*. (2016), 327–354. DOI:https://doi.org/10.1017/CBO9781139031127.025.

[5] Dickey, M.D. 2006. Game Design Narrative for Learning: Appropiating Adventure Game Design Narrative Devices of Interactive Learning Environment. *Educational Technology Research and Development*. 54, 3 (2006), 245–263. DOI:https://doi.org/10.1007/s11423-006-8806-y.

[6] DiscoveryAgents 2016. *Calgary Parks and the Agents of Nature App*.

[7] EPSG:3857: 2010. *http://wiki.openstreetmap.org/wiki/EPSG:3857*.

[8] Falconi, R.F. 2010. Usability and game design : improving the MITAR Game Editor ; Improving the MITAR Game Editor ; Usability and game design : improving the Massachusetts Institute of Technology Augmented Reality Game Editor. *MIT*. (2010).

[9] Freire, M. et al. 2016. *Game learning analytics: Learning analytics for serious games*.

[10] Hobbs, T. 2016. Why Pokémon Go is a game changer for augmented reality and marketers.

[11] Ilkovičová, Ľ. et al. 2014. Positioning in Indoor Environment using QR Codes. (2014), 117–122.

[12] Kasapakis, V. 2015. Pervasive gaming: Status, trends and design principles. *Journal of Network and Computer Applications*. 55 SRC-, (2015), 213–236. DOI:https://doi.org/http://dx.doi.org/10.1016/j.jnca.2015.05.009.

[13] Klopfer, E. 2008. *Augmented Learning: Research and Design of Mobile Educational Games*.

[14] Open Geospatial Consortium 2010. gml:AbstractGeometry.

[15] Perez-colado, I.J. et al. 2017. Integrating learning analytics into a game authoring tool. *International Conference on Web-based Learning* (2017).

[16] Perez-Colado, I.J. et al. 2017. uAdventure: The eAdventure reboot. *IEEE Education Engineering EDUCON 2017 Conference (en prensa)* (Athenas, 2017).

[17] PETERSON, B. et al. 2001. Measuring GPS signals indoors. *Proceedings of the Institute of Navigation's ION GPS-2001*. (2001), 615–624.

[18] Schneider, D. 2013. New indoor navigation technologies work where gps can't. *ieee spectrum*.

[19] Serious Game Market worth $5,448.82 Million by 2020: 2014. *http://www.marketsandmarkets.com/PressReleases/serious-game.asp*.

[20] Tile Map Service Specification: 2012. *http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification*.

[21] Torrente, J. et al. 2010. <e-Adventure>: Introducing educational games in the learning process. *IEEE EDUCON 2010 Conference* (Apr. 2010), 1121–1126.

[22] Velicanu, A. et al. 2013. INTEGRATING SERIOUS GAMES INTO E-LEARNING PLATFORMS: PRESENT AND FUTURE. *The 9th International Scientific Conference eLearning and software for Education Bucharest, April 25-26, 2013*. i (2013), 380–386.

[23] Yang, C. and Shao, H. 2015. WiFi-based indoor positioning. *IEEE Communications Magazine*. 53, 3 (Mar. 2015), 150–157. DOI:https://doi.org/10.1109/MCOM.2015.7060497.