Multimed Tools Appl (2015) 74:8137–8169
DOI 10.1007/s11042-014-2045-8

An integrated environment and development framework for social gaming using mobile devices, digital TV and Internet

Marija Punt · Milan Z. Bjelica · Vladan Zdravkovic · Nikola Teslic

Received: 2 November 2013 / Revised: 27 February 2014 / Accepted: 17 April 2014 /
Published online: 30 April 2014

Abstract The amount of digital multimedia devices in a modern day household capable of connecting to the Internet has increased dramatically over the last years, including mobile devices such as smart phones and tablets as well as digital TV sets and set-top boxes. Since these devices are readily available and allow customization through software they can be easily used to support and enhance traditional social activities in the living room. This paper presents an integrated environment of mobile devices and digital TVs connected to the Internet used as a platform for exploring both traditional and novel gaming concepts in either a single living room or across different homes connecting multiple living rooms. To create such an environment a framework was developed enabling the implementation of distributed social games, using the digital TV as a display showing game content public to all players and using the available personal mobile devices as controllers and displays showing private portions of the game. The framework also allows the innovative use of broadcast related information and social media during game play. Five different games were developed using the framework. The framework effectiveness was evaluated by comparing TV-centric games developed with and without the framework using size and complexity metrics, additionally application responsiveness was measured using a game developed without and with the framework and compared with a state-of-the-art game controller. The experience of playing the developed games was obtained by collecting and analyzing self-reported data using a questionnaire combined with additional observations from volunteers and researchers.

Keywords Social games · Mobile devices · Interactive TV · TV-centric gaming · Game development framework · Second screen

M. Punt (✉)
School of Electrical Engineering, University of Belgrade, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia
e-mail: marija.punt@etf.rs

M. Z. Bjelica · N. Teslic
Computer Engineering and Computer Communications Department, Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia

V. Zdravkovic
Sheffield Hallam University, Howard Street, Sheffield, South Yorkshire S1 1WB, UK

1 Introduction

Modern digital TVs (DTVs) that are connected to the Internet provide not only access to passively consumed broadcasted media but also allow for more interactive forms of entertainment and information retrieval. To provide an attractive and feature-rich environment for the end-user vendors are nowadays equipping DTVs with additional functionality such as browsing the Internet, parental control, games, etc. As a device shared by most household members the DTV has the potential to be a natural focal point in the living room for gathering family and friends, which makes it an ideal platform for recreating traditional social games, such as board games in digital form, to be played together around the TV on a large display.

Within the context of multiplayer games a disadvantage of DTVs is that they typically have a single shared controller (remote or keyboard), which is problematic considering that each player requires an individual controller. Currently it is common that most members in a household have access to their own personal mobile device such as a tablet or phone that can perform the role of a game controller, without the need to introduce custom hardware typically used in combination with gaming consoles. Mobile devices also offer unique features such as touch screens, motion sensors and haptic feedback that allow novel methods of interaction. An additional feature is the presence of a private screen showing information relevant only to that player that can complement the public view displayed on the TV screen visible to all players.

In this paper a new way of playing traditional social games in the living room using an integrated environment of mobile devices and DTVs connected to the Internet is presented in detail. The concept is called TV-centric gaming and it was summarily introduced in [5]. The game play considers the scenario in which the main game content is shown on the TV screen such as board, card deck, racing track etc. The mobile devices are employed as game controllers using touch screen, motion sensing functionality and haptic feedback, as well the screen on the mobile device is used as an outlet for the private portion of the game, such as item inventories or cards in a hand. The game play is enriched by the possibility to access DTV services by including broadcast related data in game play, such as an Electronic Program Guide (EPG) that can be used during game play or the games can be shown overlaid on top of the TV broadcast content. Access to the Internet allows the multiplayer concept to be extended outside of the living room to involve many other players geographically separated, some of them grouped together in the same living room sharing a single display. This allows team based play where members of the same team can be physically seated together. Additionally the integration with social media networks allows the social circle of the living room to extend to the society online. In Fig. 1 the TV-centric gaming environment is shown, involving the integration of mobile devices and DTV via the local network, using broadcast-related data and communication with the Internet.

To allow easy implementation of the TV-centric types of games a distributed game development framework was developed. There are many social gaming scenarios that can be explored in a TV-centric environment, including traditional tabletop games such as board and card games and new types of games that incorporate TV broadcast with game play elements. A single game would not be able to explore this wide range of scenarios, however there are also strong commonalities between different scenarios such as the use of secondary displays that can also be used as controllers. To avoid ad-hoc implementations that explore each scenario separately a framework was required that would not only contain all common functionality but also force a certain approach to game development creating more consistency between different implementations.

Fig. 1 The TV-centric gaming environment

The design of the framework allows different devices to be connected together easily and permits distribution of different parts of the game on multiple locations. There are two supported scenarios for connecting devices together, either by hosting the server on one of the devices located within the local area network (LAN) or by placing it on a web service in the cloud. The first allows only devices within the local area network to participate in the game, but has lower game response times. The latter allows players located in geographically separated living rooms to play together, although it results in higher game response times. The framework also defines a mechanism that allows a consistent approach to show content on multiple displays, without the need of each client having a full representation of the game world on each client, as well as making it easy to use broadcast related data and sharing results on different social networks. In recent years the market share of Android capable mobile devices, set-top boxes and DTVs has drastically risen, making Java an attractive choice for application development targeting multimedia consumer electronic devices. Therefore the Java programming language was used to develop the TV-centric gaming framework allowing the developed framework components to run unmodified on a wide variety of devices.

In parallel to the development of the framework five different games were implemented that use the framework allowing an exploration of a variety of TV-centric social gaming scenarios. To support this wide range of scenarios the framework has to be generic enough and not to focus on any specific case. Both the framework and the games implemented using the framework were evaluated. Framework evaluation was performed by measuring its effectiveness in terms of size and complexity reduction using metrics, both of which have a beneficial impact on the effort to develop and test software. Additional application responsiveness measurements were gathered to establish whether performance was acceptable and how the usage of the framework affected the performance. The TV-centric gaming experience was measured by collecting and analyzing self-reported data using a questionnaire combined with additional observations from both researchers and volunteers that played the developed games.

The rest of the paper is organized as follows. In Section 2 related work is presented. The developed TV-centric distributed game framework and its components are provided in detail in Section 3. In Section 4 an overview of the interaction between components is presented. Different types of games developed using the framework are shown in Section 5. In Section 6 the framework evaluation is presented. The results of measuring the TV-centric gaming experience are discussed in Section 7. The conclusions are outlined in Section 8.

2 Related work

Many different aspects of the topic presented in this paper were studied to some extent in previous research. The framework proposed in this paper is related to several research subareas and acts as a common ground offering a practical solution for several research themes within these subareas:

- In the area of social and interactive TV the relationship and interaction between the broadcaster and the viewer, and among viewers themselves is explored. The TV-centric gaming concept and the proposed framework intend to support interactivity which integrates seamlessly and scales up beyond classical TV use cases.
- Within the subject of second screen, ways of using mobile devices to enhance the TV viewing experience, either as a remote controller or another viewing outlet, are examined. The proposed framework utilizes and extends this concept to explore novel means of interactivity in gaming while using both screens towards a specific, well-defined purpose.
- Research in the area of human-computer interaction (HCI) in gaming has been exploring new ways of interaction between the user and the computer using gesture recognition, touch and camera sensors. The proposed framework allows for the creation of applications in which mobile devices equipped with various sensors can be used to enhance gaming experience.
- The narrow field of digitalized tabletop games deals with ways in which tabletop games can be enhanced by digital formats while keeping the interest of traditional tabletop gamers. Concepts that are explored include augmented reality principles where tangible and digital assets are used together. The proposed framework allows the utilization of common user devices, such as tablets and TVs, towards the creation of tabletop games. The framework builds upon the concept of private vs. public screen and provides the tools for facilitating development of game play across multiple screens.
- In the field of cloud gaming users from many parts of the world can play together easily over the Internet, using different devices to connect to the cloud. The proposed framework fully supports integration with the cloud and allows TV-centric games to be played at various locations around the world. Additionally, the proposed framework provides built-in functionality for connecting to existing cloud services, such as social networking platforms or metadata databases.
- The literature describes various Android game frameworks with varying capabilities. The proposed framework provides a means to complement the scope of existing frameworks towards the TV-centric gaming and an all-encompassing platform for creating this kind of games in Android.

2.1 Social and interactive TV

Television viewing used to be a social activity where friends and family would gather in the living room to watch and discuss programs together [16]. Due to ongoing developments such as an increase in the sheer quantity of programs and advances in technology such as video on demand and digital video recorders, people are less likely to gather and share the same experience at the same time [2]. As television viewing became a more individual experience, recent research tried to bring social elements back. There are a number of prototype systems that try to make interaction between remote viewers possible through various communication channels. Some of them are: ConnectTV [7], Alcatel's Amigo TV [11], AT&Labs Research's

CollaboraTV [30] and Motorola's STV [29]. All of the mentioned implementations support some type of communication, either through text or voice chats, provide users with the possibility to show emotions while watching and use buddy lists showing what their friends are currently watching. Some of them such as Amigo TV and CollaboraTV use avatars to represent remote viewers. ConnectTV provides the ability to follow buddies and to switch to the most popular channel. CollaboraTV also includes asynchronous communication by letting users leave comments at specific moments during a television show, which are then visible to other viewers when watching the same show at a later moment. Motorola's STV provides a history of what was watched either by the user or his/her buddies and what are they planning to watch.

All of the mentioned implementations focus on scenarios where viewers are physically separated trying to provide a way of making them socialize. They do not encourage friends and family to physically gather in one location around the TV to watch and to interact face-to-face. Design guidelines for the development of technology that could support social interaction around TV with both co-located and non-located friends and family members are given in [40]. One of the observations was related to gaming and stated that TV viewers are interested in having technology that could support playing informal or traditional, board and card games around the TV. The proposed development framework is essentially a follow-up in this regard and acts as an enabler for these gaming scenarios. Its purpose is to allow the creation of games that will encourage friends and family to gather in the living room, engage in face-to-face interactions, while still keeping the ability to connect to remote groups of viewers.

2.2 Second screen

Various researches indicate that mobile devices are commonly used in the living room during television viewing. A study performed by the Nielsen company [32] showed that television viewers would in parallel be engaged in other media such as email, texting, browsing, with social media access being the most common activity. The research in [12] finds that tablet owners prefer to use their devices in the living room. Although the TV set used to be the most dominant device in the living room, nowadays multiple other devices are used as well in the same space and can be incorporated in the experience of watching television.

The work done on supplementing television viewing with an external device focuses mainly on enhancing the existing functionality of the television by using a second display hosted on a mobile device to add interactivity to video content or provide supplemental information. The system in which a PDA is used for displaying a personalized EPG with additional functionality such as volume controls and channel navigation together with a comprehensive user study is presented in [13]. The study in [24] reports on the Telebuddies application which allows a mobile device to be used as a second screen for playing a quiz along with the TV show. Usage of a second screen is as well tested in a language learning service via interactive television called TAMALLE [15]. In [39] the MVC-iTV framework is introduced allowing multimedia content shown on TV to be controlled using multiple mobile devices functioning as remote controllers. In [8] mobile devices are used as a means of interacting with the TV by employing feature extraction from the video and audio to allow identification of television content and to subsequently provide the viewer with information about the program currently watched.

Current literature related to second screen explores methods for providing extra information, more advanced remote control and interactivity that is still limited to menus and quizzes. The existing research fails to combine the television and available second screens as a platform for investigating different types of game play. Additionally, most researches (MVC-iTV being

the exception) explore only the intranet use case, whereas the communication of screens via Internet still needs to be examined further. Most solutions include a specific implementation of client and server applications, with only a few using a more fundamental approach, such as offering a framework. The proposed framework incorporates the use of multiple second screens in TV-centric gaming and allows communication to take place both on the intranet and the Internet, which advances the current state of the art.

2.3 Human-computer interaction in gaming

The main focus of HCI in gaming is to provide enhanced or novel user interfaces to control games. Mobile phones equipped with compass, camera, touch screens, haptic feedback and voice control, are drawing interest from the game community by offering a great potential to function as wireless remote game controllers. In [3] a set of interviews were conducted to establish the acceptance of gestures as an interaction mode for mobile devices. Participants were asked to perform tasks using gestures that included touching of the screen with fingers, movement of the fingers across the screen, and moving the phone in several directions. A key observation is that the participants clearly favored gestures over traditional key-clicking modes of interaction.

There are a number of studies where forms of gesture recognition are implemented. A gesture recognition framework for mobile devices, called gRmobile, is introduced in [20]. A sequence action recognition framework based on state machines is developed in [1] and used in two test cases that implement a fishing and a bowling game. In [37] a generic framework is provided that uses on-board phone sensors such as cameras and accelerometers to control games and simulations on large public displays via Bluetooth. An application called MobiToss [34] allows users to take photos or capture a video by using their phone and transfer the clip onto a large public display for instant viewing by using a “throwing” gesture, furthermore the clip can be manipulated on screen by tilting the phone in different directions.

Current development frameworks related to HCI using mobile devices mainly overlooked the possibility of the mobile device to be both used as a controller and as an additional display in the game where for example private data can be shown. The proposed development framework incorporates the ability to perform motion recognition using sensor data collected from the mobile device, with the ability to apply this in novel gaming scenarios.

2.4 Digitalized tabletop games

Various research efforts are notable in the attempt to provide an enhanced gaming experience for traditional tabletop games while utilizing the advantages of digital gaming worlds. The latest research employs technologies such as projectors, visual recognition, RFID, touch surfaces and second screen displays to provide digital inputs and outputs for tabletop games. The approaches used include applying augmented reality to existing physical objects (card or board pawns), or using virtual items on top of physical reality (e.g. poker surface with virtual chips or cards).

In [33] a card game is developed using an RFID system connected to a desktop PC, a set of PDAs, combined with a standard 52 card deck, where each card is equipped with an RFID tag. The server application, running on a PC, is maintaining the game state using the RFID system. The client applications are running on PDAs classifying the player’s moves as being correct or not, using a visual smiley face indicator. The STARS platform presented in [25] is designed to

support classical board games with the use of various multimedia-devices. The STARS setup allows developers to create complex game scenarios which can have both collaborative and competitive elements in one game. It provides both wall and digital tabletop displays. Setup components include a touch sensitive plasma display that acts as the game board which is coupled with a camera capturing the setup from the top. The camera allows the system to detect and identify game pawns on the interactive screen. In addition, the table includes RFID readers which in combination with RFID tagged objects can be used to save and load different scenarios and games. A digital poker card game, called Poker Surface, which combines mobile phone gestures and multitouch tabletop interactions is developed in [14]. Accelerometer-enabled phones are connected via Bluetooth to the table and are used both as gesture inputs and for private displays of cards in hand. Preliminary results show that the users preferred the phone for interaction compared to multitouch interaction using the table. In [27] a study is conducted exploring the use of a phone as a controller for dial manipulation tasks, comparing three different forms of interaction: direct touch, using the phone as a general-purpose tangible controller on a tabletop touch screen computer and manipulating the dial directly on the phone's screen. It was found that using the phone as a general-purpose active tangible on the tabletop offers no improvements over direct touch and that the best place to perform the interaction is directly on the phone screen.

Most of the research works present appealing usage scenarios in which actions are simulated and the experience is as close to real life as possible, proving the feasibility of the approach. However they require custom hardware which is expensive and hard to setup in an actual living room or hard to be moved around. Also, readily available mobile devices are considered an adequate if not more preferable means of interaction compared to large touch screen tabletops. The proposed framework acts as a turn-key solution for the digitalization of tabletop games, in which the existing hardware present in a living room is used to achieve a traditional tabletop gaming experience.

2.5 Cloud gaming

Cloud computing and cloud gaming are diverse and intense areas of research in recent years. The main concept of research in this area is to allow all the devices to use a standard API in the cloud (on the Internet), to perform tasks commonly reserved to be implemented on the devices themselves. This way all the logic and performance-dependent operations are offloaded to Internet servers. In the realm of mobile devices, the Cuckoo framework [21] suggests offloading performance-intensive smartphone tasks to the cloud. Cloud gaming is the field where techniques used in cloud computing are applied to gaming. For example tasks can be partially offloaded by placing the game logic in the cloud, but leaving the rendering task to the client device [41]. A similar approach is used in the proposed development framework that can also be configured to execute game logic in its cloud service.

Another concept explored in the literature is the ability to interact with various third party cloud API-s, such as social network APIs, search APIs, geolocation APIs, metadata providers, etc. An application development framework OpenSocial is proposed, to enable the creation of applications that are interoperable within the context of different social networks [18]. The main advantage of the proposed framework over the existing solutions is that it provides the fusion between broadcast and broadband worlds. It is made possible, through a set of APIs, that the application takes metadata from broadcast streams (e.g. EPG) and pushes it to one or more social networks, or uses it to retrieve additional metadata from online databases, which is to the best of our knowledge a novel solution.

2.6 Android game frameworks

Existing Android game frameworks are mainly focused on game development challenges such as 2D/3D scene rendering and level scripting. Among the 2D game development frameworks currently existing for Android, the following open-source frameworks are notable [42]: AndEngine, LibGDX and Cocos2D-X. AndEngine [35] is a broad 2D game engine facilitating operations such as resource management, handling of game entities, sprites, text, meshes, etc. The engine allows positioning of entities in the scene, using object modifiers and custom programming employing OpenGL capabilities, applying physics to the game and working with multi-touch events and gestures. LibGDX [22] is a framework which supports Particle Systems, Tile Maps, Box2D, 3D objects, Fonts, UI elements etc. This framework acts as a wrapper around OpenGL with a specific focus on game performance and optimization. Cocos2D-X [19] is an open source 2D game development framework branched to many different target platforms, with cross-platform development at its core. This framework also handles tiles, sprites, scenes and common GUI elements.

The intention of the proposed game development framework is not to compete or surpass the capabilities offered by the rendering modules within the frameworks already available. The proposed approach is different in that it focuses on supporting second display functionality without custom implementation. It also has explicit support for different game categories, and allows for interaction with both broadcast and cloud services, which is not incorporated in the evaluated frameworks.

Overall, the proposed development framework makes an offset to the current state of the art by combining the use of the television as a gathering point for friends and family, while at the same time allowing interaction with remote groups of people. The framework incorporates the use of social networking services as well as cloud based computing. It presents a significant contribution to digitalization of traditional tabletop games, using existing ecosystems of devices, potentially with the inclusion of broadcast content. The framework also enables the use of mobile devices in two roles, as a second display providing private content, and as a game controller utilizing the available motion and touch sensors.

3 TV-centric game development framework

The developed framework consists of multiple components: Game Logic, Display, Communication, Broadcast, Social Media and Motion Sensing. The core of the framework is the Game Logic component that defines the fundamental behavior of games and also defines a hierarchy of different game categories, each one specifying additional behavior and building blocks. The ability to render the game world on the screens of all devices involved in the game play is provided by the Display component. The Communication component is responsible for all data exchange between devices and services participating in the same game. The main purpose of the Broadcast component is to wrap the already existing Java API interface developed for the integration of DTV services in modern DTVs and set-top boxes [38]. The Social Media component allows a game to interface with common social media platforms. The Motion Sensing component is only used on mobile devices and provides easy to use motion sensing recognition functionality built on top of the accelerometer and gyroscope sensors. A block diagram of the framework and its position within the software stack in embedded multimedia devices with DTV integration and support for Java is shown in Fig. 2. In the following text each framework component will be described in more detail.

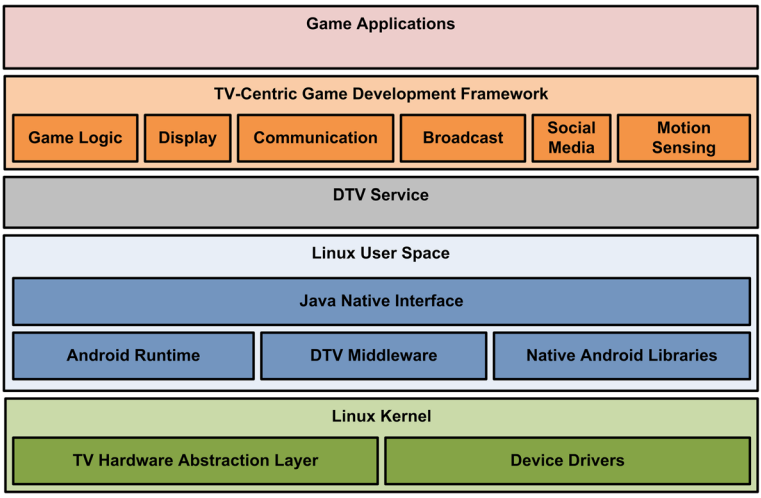


Fig. 2 TV-centric game development framework block diagram

3.1 Game logic component

The Game Logic component specifies a number of generic abstract classes that form the logical building blocks to construct a game. The UML diagram in Fig. 3 shows the main game

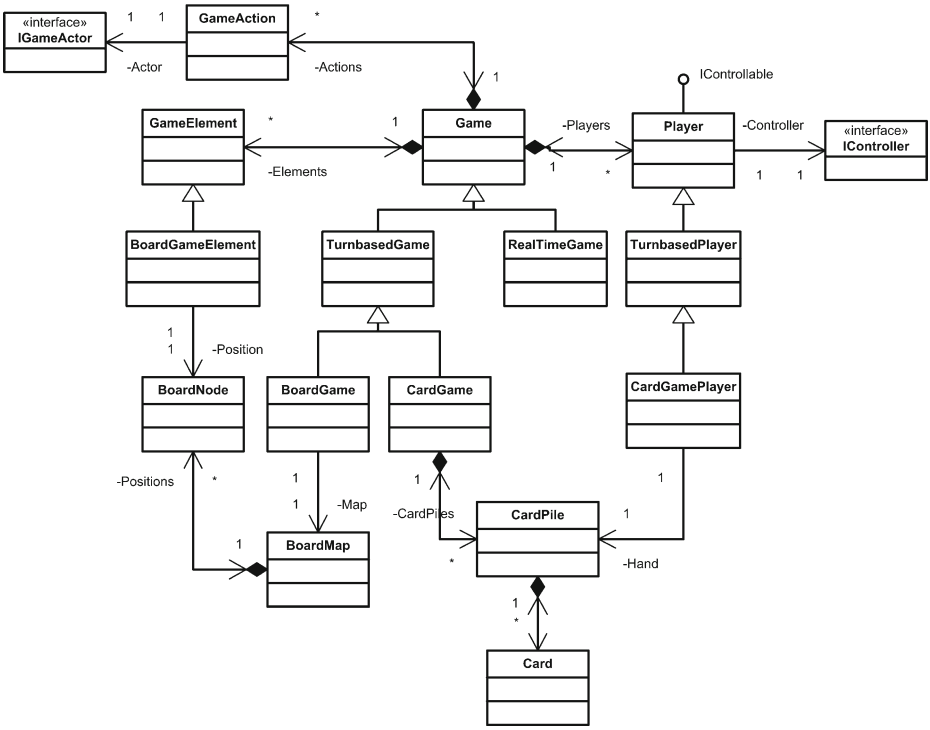


Fig. 3 UML diagram of base classes in the Game Logic component

logic classes and their relationships. The Game class is the abstract base class that defines a two dimensional game world and the basic rules that govern the behavior of games. Changes to the game state are calculated in iterations that are typically configured by the developer to represent a small unit of time thereby creating the illusion of continuous progression of time. The Game class contains references to three types of objects: Player instances, GameElement instances and GameAction instances. The Game class instance and related instances are typically placed on the server.

Each instance of a Player class represents a real-world player with associated name, score, etc. The Player object has the ability to establish two-way communication with the controller logic implemented in the client application running on a mobile device. The controller logic collects all forms of player input, such as touch or motion events and sends this information to the Player instance. The Player instance can also send messages to the controller logic, which for example can be used to trigger haptic feedback on the mobile device to indicate a player's turn. Each instance of a GameElement class defines a game object with associated coordinates and dimensions in the game world. Each instance of a GameAction represents an action that affects the game state for an associated duration; an example would be the movement of a pawn from one position to the next on a board. While active the GameAction instance will modify the game state each iteration. Once the action is finished it will invoke a callback method on an instance of a class implementing the IGameActor interface to signal the end and outcome of the action. Any concrete class can implement this interface.

The Game, Player and GameElement base classes are extended through class inheritance to form a hierarchy of game categories that are representative for the type of games targeted by the TV-centric approach. The TurnBasedGame and TurnbasedPlayer classes are abstract subclasses of the Game and Player classes respectively and together introduce turn-based game play and rounds. Once a turn-based game starts the first round commences, which will give the turn to the first TurnbasedPlayer instance registered to the game. Once the player ends its turn, the turn is awarded to the next TurnbasedPlayer instance, if no more players are present the next round starts and the process starts over again. This contrasts with the RealTimeGame class which allows all players to simultaneously control the game-play. The BoardGame class is an abstract subclass of the TurnbasedGame used to implement board games. Each board game will include a reference to a BoardMap instance that is used as a representation of a board with discrete positions and allowed movements between them. Individual positions on the board are represented as instances of the BoardNode class. The BoardMap class implements common functionality such as finding paths between positions. The BoardGameElement class is an abstract subclass of the GameElement class used to represent game objects that can be placed on discrete positions on the board map. It is used exclusively in combination with the BoardGame class. Typical examples of BoardGameElement instances are pawns that can be moved across the board. The CardGame class is an abstract subclass of the TurnbasedGame class used to implement card games. The CardPile class is used to represent collections of cards, for example the player's hand, or decks of cards that are on the table.

3.2 Display component

The Game Logic is responsible for updating the Display component using the minimal amount of data required to render the game world. For each iteration the game calculates the changes relevant to a display and sends it to the Display component. A single game will typically have multiple instances of the Display component showing the game world on different devices. Each instance of the Display component can be

configured by the Game Logic component to show or hide portions of the game world. For example the TV shows the game world and elements public to all players. All players have their own view of the game world displayed on their mobile devices, which includes private information relevant only to each individual player.

Each instance of a Display component is keeping a list of objects that can be drawn. These objects are instances of the Drawable class containing a subset of GameElement attributes that are relevant for drawing. Some of these attributes are mutable (such as position and state) and other attributes are immutable (such as the identifier of the GameElement object it represents). Drawable instances are originally produced by the game logic, but are consumed by the Display component. In each iteration of the game, changes occur to the state of its GameElement instances. These changes should also be reflected in the state of the Drawable instances that are used by the Display component. One approach to keep the Display component updated would be to send all Drawable instances to a display for each iteration to make sure they have the information required to render the game world correctly. Another more economic approach used in the TV-centric framework is to inform the Display component of changes that occurred to the list of Drawable instances that should be rendered, excluding a possibly large set of Drawable instances that remained unchanged between two iterations. The types of changes that can occur in the game world and are relevant for the Display are: addition, modification and removal of GameElement instances. These types of changes correspond with the addition, modification and removal of Drawable instances on the Display component instance. To add a Drawable to a Display the entire object must be provided. To describe changes to an existing Drawable it is sufficient to provide only attributes that can be modified. The DrawableChange class contains only attributes of the Drawable class that are mutable. The Drawable class implements a method that allows its mutable attributes to be updated with an instance of the DrawableChange class. Removal of Drawable instances can be achieved by providing the identifiers of the GameElement instances that were removed. The changes to the state of Drawable instances on a display are stored as an instance of an IterationChange class. Instances of this class can be serialized and transmitted as JSON messages to Display components hosted on other devices, or directly passed to a Display component if present on the same device. The display will use the information in the IterationChange instance to update its own list of Drawable instances. For each game iteration one IterationChange instance is produced.

Drawable instances that are provided by the game logic do not contain graphical assets themselves, but do refer to assets by their unique identifiers. The assets themselves are initialized by the application hosting the Display component and are at that point scaled to match the resolution of the physical display that will show the graphics. The graphics are cached to ensure fast retrieval during rendering. The rendering of the graphics is done using standard Android drawing primitives allowing graphics to be rendered as overlaid graphics on top of existing video controls, which was used to combine broadcasted video streams and game play. The rendering mechanism also includes the ability to draw non-transparent background in cases where the Broadcast component is not used. The Display component can show either the entire game world by scaling it down to fit on the available space on the screen, or it can show a subset of the game world and pan around and track objects as they move through the game world. The first mode is typically used on a public display, while the latter is used on smaller private displays. It is also possible for a game to define that a game element is anchored to a fixed position within the viewport itself, so it does not move even if the camera viewpoint does, an example of an anchored game element would be the player's

score. The Display component registers touch events, provides information on the game world coordinate that was touched and lists the identifiers of all game elements that occupy that coordinate.

3.2.1 Scope mechanism

If a game uses multiple displays and requires that each display show a different same set of elements then the scope mechanism is used as a means of declaring visibility of objects on different displays. For example a player's private display might show items that should remain hidden from others. When creating an instance of a GameElement class the game implementation must declare which scope it belongs too. An element can belong only to a single scope, but as many scopes as needed can be defined. Each display can subscribe to one or more scopes, which will determine the set of objects that a display will be informed about and accordingly display. Before an IterationChange instance is sent to each display a filter is applied based on the scopes that a display is subscribed to, the resulting IterationChange object will only contain additions, modifications and removals of relevant Drawable instances.

The usage of scopes will be presented through an example of a multi-player board game with four players and two teams consisting of two players per team. The game involves moving pawns on the board by throwing dice and placing mines on the board that are invisible to members of the other team, but not to members of the same team. Five displays will be used in total, one private display for each player and a single public display. In Table 1 the eight possible scopes are presented: *Universal*, *Player1 Private*, *Player2 Private*, *Player3 Private*, *Player4 Private*, *Team1 Private*, *Team2 Private* and *Public Only* scope. For each scope, examples of game elements that belong to that scope are listed. Pawns and explosions belong to the *Universal* scope. A player's inventory belongs only to the appropriate player's private scope. Mines placed by the members of a team belong to the appropriate team's private scope. Scores belong to the *Public only* scope. For each of the five existing displays, the scopes they are subscribed to and the game elements that are accordingly visible on that display are listed in Table 2. For example every display is subscribed to scope *Universal*, thus Pawns and Explosions will be shown on all displays. Only the displays of player 1 and player 2 are subscribed to the *Team1 Private* scope, which means that mines placed by one of them will be shown on those two displays only.

Table 1 Scopes used in multi-player board game example

Scopes	Game element examples
Universal	Pawns, Explosions
Player1 private	Player1 Inventory
Player2 private	Player2 Inventory
Player3 private	Player3 Inventory
Player4 private	Player4 Inventory
Team1 private	Team1 Mines
Team2 private	Team2 Mines
Public only	Scores

Table 2 Displays used in multi-player board game example, with their scope subscriptions and resulting visible game elements

Display	Subscribed to scopes	Game elements visible
Public display	{Universal, Public Only}	{Pawns, Explosions, Scores}
Player1 display	{Universal, Player1 Private, Team1 Private}	{Pawns, Explosions, Player1 Inventory, Team1 Mines}
Player2 display	{Universal, Player2 Private, Team1 Private}	{Pawns, Explosions, Player2 Inventory, Team1 Mines}
Player3 display	{Universal, Player3 Private, Team2 Private}	{Pawns, Explosions, Player3 Inventory, Team2 Mines}
Player4 display	{Universal, Player4 Private, Team2 Private}	{Pawns, Explosions, Player4 Inventory, Team2 Mines}

3.3 Communication component

The Communication component supports the development of multiplayer games played within either a local area network or via the cloud enabling the game to be hosted as a cloud service. Communication within the local area network is implemented using TCP/IP sockets supplemented with an automatic server discovery mechanism that can be initiated by the client using a UDP multicast message, to which a server within the local area network can respond. Once the client is aware of all the servers on the network a TCP/IP socket connection is opened to the chosen server. After the connection is established, data can be sent between client and server.

Communication with web services hosted in the cloud is implemented using the HTTP protocol. The component provides the developer with the means to easily manage HTTP requests and responses between client and server. Furthermore the component supports the development of games hosted in the cloud by including a game session management mechanism taking care of the life cycle of each game session on the server. This mechanism includes the creation and termination of game sessions and the dynamic tracking of each connected client application within a session.

A client application can initiate either a public or private game session. In both cases a game session key is returned. This key allows a separate client application used only to display the game on the TV screen to connect to the same game session. In the case of private sessions the key can be distributed to a group of friends, allowing them to play together. Anyone can join public sessions without the explicit distribution of the session key. Game sessions are fully independent of each other, enabling the web service to host many simultaneous game sessions each potentially belonging to a different type of game. Communication between the devices and the web services hosted in the cloud is shown in Fig. 4.

3.4 Broadcast component

The Broadcast component wraps the DTV Java service to enable games built using the framework to take advantage of the broadcast related functionality and data. This DTV service is initialized upon the startup of the set-top box. Any game application communicating with the service obtains an instance to a class implementing the IDTVManager interface, which gives access to the DTV API. Through the IDTVManager, the game application has the ability to fetch a specific interface towards a desired piece of DTV functionality. The main services that can be used in game play are retrieving the channel name, information about what is now

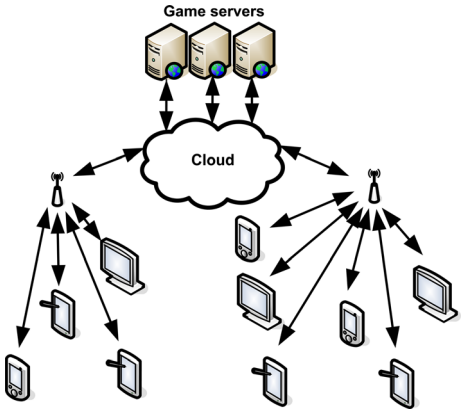


Fig. 4 Communication between the devices and the web services hosted in the cloud

playing, what will play next, changing the channel, getting or setting the volume level. In order to ensure smooth game play the wrapper provides non-blocking methods that employ deferred callbacks to return the outcome of the request.

The DTV service is coupled with the DTV middleware running as native C code (Fig. 2). This coupling is performed through the Java Native Interface (JNI) mechanism, which enables Java-based services to execute routines written in C. On the other hand, all the events generated by the DTV middleware, are reported back to the DTV service through JNI, by calling a Java method designated for the event reception. This method parses the event report and passes it on to an appropriate DTV Java object.

The DTV middleware is in charge of monitoring the transport stream, extracting specific service information tables and providing easy-to-access information such as service lists, and event schedules. The DTV middleware also has access to specific hardware components within a set-top box, therefore allowing clients for example to tune to a specific frequency or trigger the recording process. The DTV hardware components present in the set-top box are abstracted from the middleware by a dedicated hardware abstraction layer API.

3.5 Social media component

The Social Media component has support for accessing the web services for the following social media platforms: Facebook and Twitter, and following metadata databases: Internet Movie Database (IMDb) and The Movie Database (TMDb). Access to each web service is encapsulated in wrapper logic that provides a uniform set of classes to the developer to access each web service without having to deal with the communication protocol itself and the session management. The social media web services accessed by the component can have large response times, sometimes measured in seconds. To prevent the Social Media component from interrupting the game-play the wrapper logic handles calls without blocking and in a separate thread will handle the request. Once the request is concluded the wrapper will callback to indicate the outcome of the call.

The following scenarios are supported by the Social Media component: the component can be used to post to a player's wall in Facebook or Twitter or alternatively use a profile defined for the game itself and post to its wall. Pre-defined messages are used that can include names of players for announcing the winner and taunt the loser, or challenge friends to play. If the

game uses the Broadcast component to determine what show is being watched during game play, this information can be included in the pre-defined messages. Posts also include screenshots collected during game play, which show key-moments such as a player crossing the finish line. The Social Media component provides access to both the IMDb and TMDb service, which are used to provide supplemental information about movies and television shows. The TMDb service allows users to post ratings to movies, which can be used in combination with the Broadcast component to construct a game allowing a player to rate a currently watched movie through some game play mechanism.

3.6 Motion sensing component

The Motion Sensing component provides motion sensing recognition functionality built on top of the accelerometer and gyroscope sensors present in mobile devices. This functionality supports hand gesture recognition for actions such as shaking or swinging. These actions trigger an event handler in case an action has been detected. Support for continuous monitoring is added as well to simplify control schemes such as using the mobile device as a steering wheel or a balancing platform.

4 Information exchange between framework components

To illustrate the information flow between components an example board game, played over the cloud by two players, in a specific state was presented. On the main TV screen the entire game board is visible containing one blue pawn belonging to the blue player and one green pawn belonging to the green player. Each of the players has a mobile device on which a screen shows a partial view of the game world centered on the player's pawn. The blue player has rolled a two and is now presented with blue highlights indicating possible positions where the blue pawn can be moved. The player can touch the location to where the pawn should move, making the display an integral part of the control scheme of the board game. In Fig. 5 the information flow between framework components using the example board game is shown. Four applications: Blue Player Game Client, Green Player Game Client, Public Display Client, and Game Web Service are presented in the diagram separated by dotted lines.

The Game Web Service application implements the game logic consisting of a single instance of the BoardGame class and two registered instances of the Player class. The Player Game Client applications are used to control the Player instances and are identical for both blue and green player. Each application contains an instance of the Display component that is configured to show a partial view of the game world. It also contains an instance of the Motion Sensing and Social Media components for detecting motion events and posting the results to social networks. The controller logic consists of custom code responsible for collecting user inputs, exchanging information with the components present in the client application as well as exchanging information with the associated Player instance present in the remote game logic. The Communication component is shown as a transparent layer.

The display on the blue player's device shows a partial view of the game world, where the camera is centered on the blue pawn. The highlighted positions are GameElement instances produced by the game for the duration of the turn, after which the game will remove them. The highlighted positions are assigned to the private scope of the blue player therefore they do not appear on any other display. Touching these highlighted positions will inform the controller logic, which in turn will inform the Player object resulting in the initiation of the pawn movement. The display on the green player's device shows a partial view of the game world

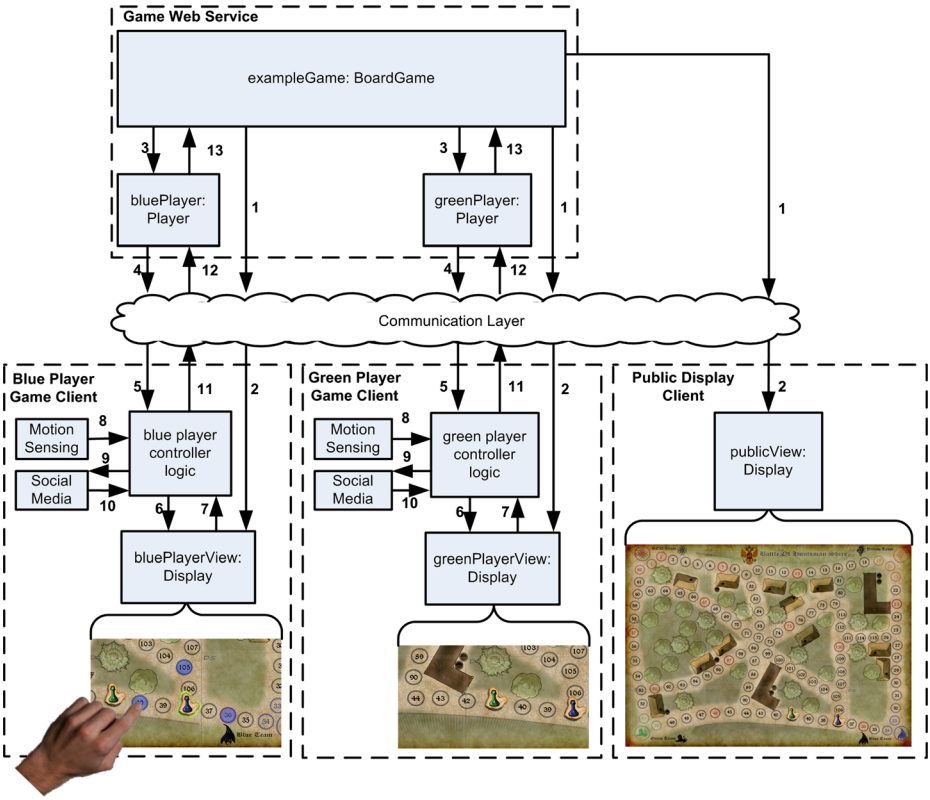


Fig. 5 Information flow between framework components

centered on the green pawn. The blue pawn is visible to the right and highlighted positions are not visible. The public display shows the entire game world containing both the blue and the green pawns, without any highlighted positions shown.

In Fig. 5 the numbered lines between the components are indicating exchange of the information in the following manner:

1. Each iteration the BoardGame instance produces an IterationChange instance. The scoping mechanism is applied to this instance to produce filtered IterationChange instances that are passed to the communication layer, where they are serialized and transmitted.
2. The communication layer deserializes the IterationChange instance and passes it on to an instance of the Display component in order to render the game world.
3. The BoardGame instance informs the Player instance of changes in the game state, for example when the player receives its turn.
4. The instances of the Player class send game state notifications to the controller logic that affect the behavior of the game logic. For example a Player instance receives a turn from the game, it then sends a message to the controller logic to trigger haptic feedback. Another example is when the Player instance sends a command to the controller to center the camera on the pawn to be moved. These messages are passed to the communication layer, where they are serialized and transmitted.

5. The communication layer deserializes the Player messages on the client side and passes them on to the controller logic, where the appropriate action is taken.
6. The controller logic passes information to the Display component to control the camera, for example when setting the center of the camera to point to a different pawn.
7. The Display component forwards the detected touch-events to the controller logic.
8. The Motion Sensing component forwards the detected motion-events to the controller logic.
9. The controller logic sends game results to the Social Media component to report to the outside world.
10. The Social Media component calls back the controller logic to indicate that the post was successfully made.
11. The controller logic sends control messages to the corresponding Player instance through the communication layer, where they are serialized and transmitted.
12. The communication layer deserializes the messages on the server side and passes them on to the instance of the Player class, where the appropriate changes to the game state are made.
13. The Player instance will interpret the control messages it received and change the state of the game if required.

The lines in Fig. 5 pointing to or from the communication layer have an additional step implemented by intermediate relay and proxy classes. The purpose of these intermediate classes is to hide the presence of a communication layer and decouple framework components to allow them to be used in a distributed environment. In situations, such as the presented example, where instances of the Display component are not present on the same device as the Game instance, the IterationChange instances must be transported via intermediate classes. An instance of the DisplayUpdateRelay class collects the IterationChange instances, serializes them and sends them to all connected clients hosting an instance of a Display. An instance of the DisplayUpdateProxy class receives the display update messages, deserializes them and provides them to the Display instance. The IDisplayUpdateProvider interface defines a producer of IterationChange instances. This interface is implemented by both the Game class and by the DisplayUpdateProxy class. Information exchanged between the Game instance and the Display component is shown in Fig. 6a.

The user inputs of a game player will be collected by the controller logic and forwarded as command messages to the Player. The Player class implements the IControllable interface indicating that it can accept command messages from controller logic. The controller logic implements the IController interface indicating that it can accept messages from a Player instance. Player instances are typically not present on the same device as the controller logic. Therefore they exchange information between themselves through intermediate proxy objects that communicate with each other over the network using the Communication component. Information exchanged between a Player instance and controller logic is shown in Fig. 6b.

5 Developed applications

Using the TV-centric distributed game framework two turn-based games (one board and one card game) were developed to be used either over a local area network or via the cloud. Three real-time micro games were developed intended to be played over the local area network during a TV broadcast. Each game is distributed over multiple applications using framework components.

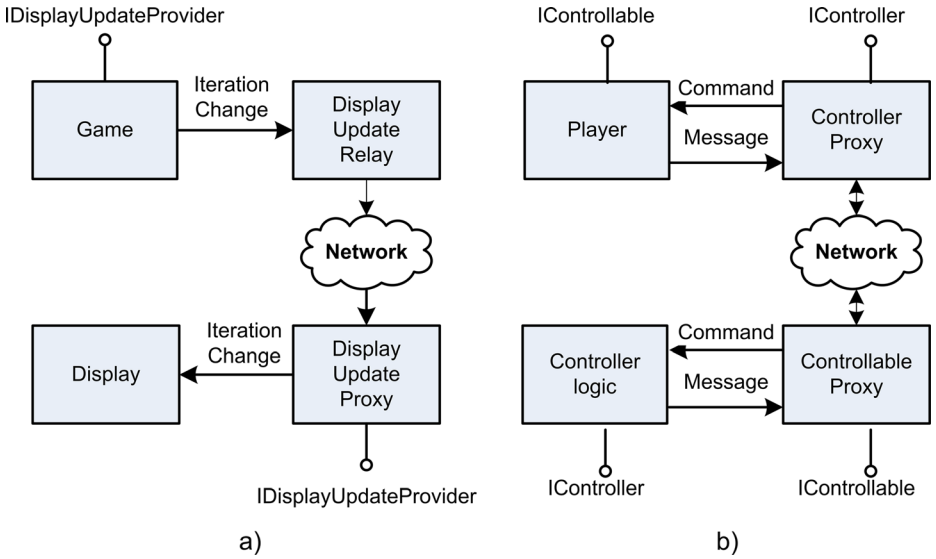


Fig. 6 **a** Information exchange between Game and Display instances **b** Information exchange between a **Player** instances and controller logic

The implemented board game is the "Dice of Blood" game with a medieval thematic. Up to four players act as enemies moving their pawns across the board with the goal to destroy one another. The board is shown on a TV screen, whereas mobile devices are used for pawn control, rolling the dice and deploying items (sword, crossbow, land mine etc.) which were gathered by the players and stored in their private inventory, visible only from a mobile device. The developed game is shown in Fig. 7.

The distribution of the framework components used for development of the "Dice of Blood" board game to be played via the cloud is shown in Fig. 8. The game web service

Fig. 7 TV-centric "Dice of Blood" game

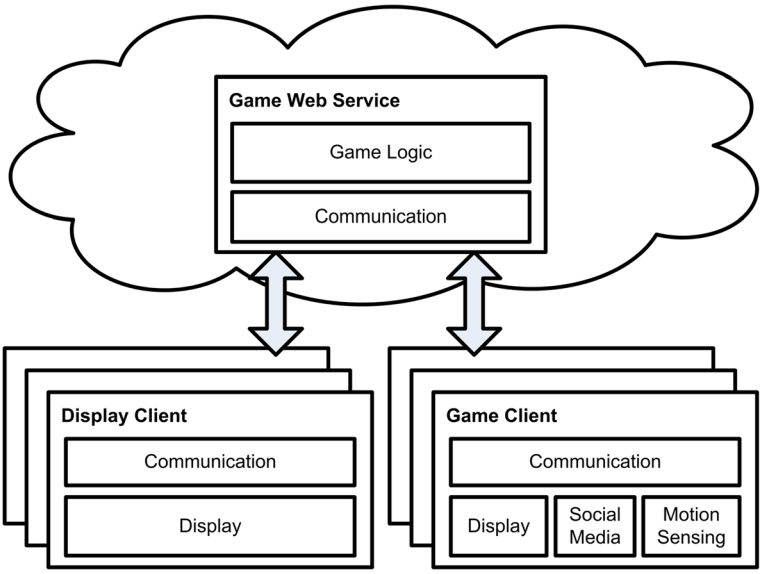


Fig. 8 Distribution of the framework components in the "Dice of Blood" game played via the cloud

hosts the only instance of the Game Logic component. There are two client applications: the display client and the game client. The display client hosts an instance of the Display component to be used on a public TV screen. The game client includes an instance of the Display component to show the player's private view of the game on mobile devices. The Motion Sensing component is used to allow the player to virtually shake and throw the dice. The Social Media component is used on the game client to report game results to the outside world. The Communication component is present in all applications allowing them to exchange data.

The implemented card game is the traditional "Crazy Eights" game. Contents of the table with a deck of cards and player scores are shown on the TV screen, whereas the cards in each of the players' hands are displayed on the mobile device. Users swipe out cards to the "table" while they are playing with the goal to use up all their cards as soon as possible. The distribution of framework components is identical to the board game, with the exception that the Motion Sensing component was not used.

The first micro game is an egg-and-spoon race between two players. The race is finished when one of the players reaches the finish line with the egg still on the spoon, or at the moment one of the players drops the egg. The TV screen shows the racing track and the runners. On each mobile device the egg on the spoon and two buttons are displayed. The player has to rapidly press two buttons alternately to propel the runner, whilst balancing an egg on a spoon. The developed "Egg-and-Spoon" game is shown in Fig. 9.

The distribution of the framework components used for the development of the "Egg-and-Spoon" game running over the local area network is shown in Fig. 10. There are two separate applications: the game server and the game client. The game server hosts both the Game Logic component executing the running game and the Display component that is used to show the track and the runners at the bottom of the TV screen. The Broadcast component is used to have control over the TV broadcast that is shown as the background of the game. The Social Media component is used to announce the winner once a game is finished to the social networking

Fig. 9 "Egg-and-Spoon" game play during a TV show

page of the game, stating the names of the players, scores and the TV context (for example, "Joe defeated Mike while watching Show A on TV Station B"). The game client also hosts an instance of the Game Logic component used to implement the balancing game on the mobile device, the outcome of that internal game controls the main game executed on the server. The Motion Sensing component is used by the balancing game to balance the egg on the spoon. The Display component on the game client shows the client-side game world consisting of the egg, spoon and buttons.

In the second micro game called "Flyswatter" flies start buzzing across the screen and players use their phones as swatters to kill as much flies as possible within a given time. The distribution of framework components is similar to the "Egg-and-Spoon" game, with the exception that the Motion Sensing component is used to recognize swatting gestures and there is no instance of the Game Logic component present in the game client application.

The third micro game called "Tomato Rating" allows players to throw tomatoes at TV programs currently showing that they are not enjoying. The component distribution is identical to the "Flyswatter" game. The Motion Sensing component is used on the mobile devices to recognize a throwing gesture and according to the throwing power the game logic calculates the number of tomatoes (one to five) to be animated on the public screen as flying towards the

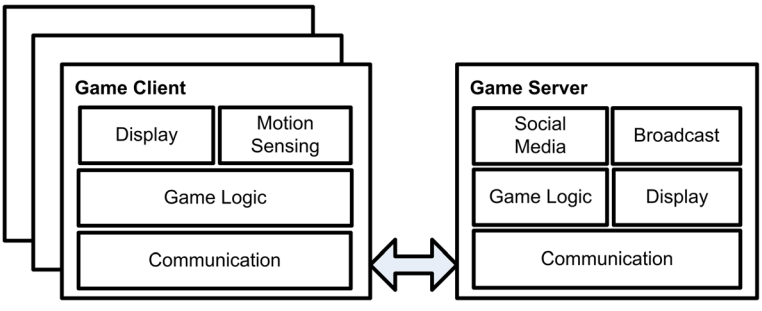


Fig. 10 Distribution of the framework components in the "Egg-and-Spoon" race game played over the local area network

show and splashing. The show rating is calculated as inversely proportional to the throwing power and posted to social networks and public movie databases using the Social Media component.

6 Framework evaluation

The framework was evaluated to give an indication of its effectiveness in terms of reducing development effort by comparing metrics collected from games developed without the framework and those that were developed using the framework. Additionally application responsiveness when using gesture recognition was measured for a game developed both with and without the framework, and compared with similar data available in the literature.

6.1 Framework effectiveness

Before the development of the framework began, earlier prototypes of the "Flyswatter" and "Dice of Blood" games were developed providing an opportunity for comparison with the same games developed using the framework. The open source LocMetrics [23] tool was used to objectively compare the size and complexity of the two versions of the games without and with the framework. The metrics used were the physical Source Lines of Code (SLOC-P) [31], the cyclomatic complexity (MVG) [28] and complexity density [17]. The definition of SLOC-P as used in LocMetrics is the total lines of source code minus blank lines and comment lines, giving an indication of programming time and effort and is an essential input for most cost estimation models [6]. Cyclomatic complexity is defined as the number of linearly-independent paths through a program module indicating the amount of tests required to evaluate the correctness of a module. The complexity density is defined as the ratio of the cyclomatic complexity of a module to its length in SLOC-P representing the normalized complexity of a section of code and is an indicator of maintenance task difficulty.

Since the prototypes offer a subset of the functionality present in the framework version of the same games, care was taken to ensure a fair comparison. For each game implementation the total Java source code was collected from both server and client applications and grouped together. For the version of the games that use the framework the code was split into new custom code and the framework code itself, for the prototype version all code was considered old custom code. The amount of framework code used in each game differed and only framework classes that were referred to by the game implementation were included in the comparison. For example the "Flyswatter" game did not use any of the board game related functionality, and therefore these classes were not included in the framework category for this game comparison. Another important criterion for inclusion was whether the functionality implemented in the code was present in both versions of the game. The prototype version of the "Dice of Blood" game did not include functionality corresponding to the Motion Sensing component, also it only supported the local area network version of the game play. Therefore the Motion Sensing component and code related to hosting a game on a web server were excluded from comparison. For the prototype version of the "Flyswatter" game the graphics were not rendered on top of a television broadcast, but instead were rendered on top of a pre-defined video stream without any access to the full broadcast related functionality. Therefore the Broadcast component was excluded from the comparison.

Subsequently the code was categorized by components, used in both versions without and with the framework, as implemented in the framework: Communication, Game Logic, Display, Social Media, Motion Sensing. For the new custom code the categorization was made by

evaluating which framework component was used. For the old custom code the evaluation was done by looking at the responsibilities for a given section of code and finding the category that most closely matches it. Ideally a single source file would fall within one of the categories, but especially in the old custom code mixed categories occurred. In these cases classification was made on a method per method basis, since any further division could affect cyclomatic complexity. The code related to initializing an application within Android and the use of standard GUI components, was categorized under the Display component, since this code was most closely related to the responsibilities implemented in the Display component.

Table 3 shows the collected metrics for the "Flyswatter" game and Table 4 shows the same data for "Dice of Blood" game. Each table column contains a single category based on game framework components, with a total sum column at the end. The rows are organized into nested sections, the first level divides the table into the game implementation without the framework and into the game implementation using the framework. The second level only divides the game implementation using the framework into three parts: new custom code, framework only code and the total code. For the game implementation without framework all code is indicated as old custom code. The third level then indicates the metrics used: SLOC-P, cyclomatic complexity and complexity density.

Figure 11 presents SLOC-P and cyclomatic complexity respectively for both "Flyswatter" and "Dice of Blood" game developed without and with framework. Since the cyclomatic complexity is highly correlated to SLOC-P the comparison of complexity of the "Flyswatter" and "Dice of Blood" game without and with the framework show a similar picture as the comparison using only SLOC-P. The overall size and the cyclomatic complexity of the new custom code are smaller than the old custom code in both game examples. Both the SLOC-P and cyclomatic complexity values for the "Fly Swatter" new custom code are 53 % that of the old custom code. The "Dice of Blood" new custom code is 47 % the size of the old custom code, while the cyclomatic complexity for the new custom code is 39 % compared to the cyclomatic complexity of the old custom code. When combining the new custom code together with the framework code used, the total size of the "Flyswatter" game is 151 % the size of the old custom code, when doing the same for the cyclomatic complexity the new custom code is 179 % that of the cyclomatic complexity for the old custom code. For the "Dice

Table 3 SLOC-P, cyclomatic complexity and complexity density collected for the "Flyswatter" game implementations without and with framework

			Communication	Game logic	Display	Motion sensing	Social media	Total
Flyswatter without framework	Old custom	SLOC-P	953	664	794	203	1,025	3,639
		MVG	81	74	37	21	111	324
		Density	0.085	0.111	0.047	0.103	0.108	0.089
Flyswatter with framework	New custom	SLOC-P	468	754	619	59	26	1,926
		MVG	46	95	22	7	3	173
		Density	0.098	0.126	0.036	0.119	0.115	0.090
	Framework	SLOC-P	618	512	1,156	252	1,045	3,583
		MVG	39	73	149	30	115	406
		Density	0.063	0.143	0.129	0.119	0.110	0.113
	New custom + framework	SLOC-P	1,086	1,266	1,775	311	1,071	5,509
		MVG	85	168	171	37	118	579
		Density	0.078	0.133	0.096	0.119	0.110	0.105

Table 4 SLOC-P, cyclomatic complexity and complexity density collected for the "Dice of Blood" game implementations without and with framework

			Communication	Game logic	Display	Social media	Total
Dice of blood without framework	Old custom	SLOC-P	1,192	4,850	4,444	1,204	11,690
		MVG	120	638	554	132	1,444
		Density	0.101	0.132	0.125	0.110	0.124
Dice of blood with framework	New custom	SLOC-P	305	3,687	1,460	31	5,483
		MVG	16	490	49	3	558
		Density	0.052	0.133	0.034	0.097	0.102
	Framework	SLOC-P	618	858	1,156	1,045	3,677
		MVG	39	117	149	115	420
		Density	0.063	0.136	0.129	0.110	0.114
	New custom + framework	SLOC-P	923	4,545	2,616	1,076	9,160
		MVG	55	607	198	118	978
		Density	0.060	0.134	0.076	0.110	0.107

of Blood" game the combined size of new custom code and framework code used is 78 % the size of the old custom code, while the cyclomatic complexity for the new custom code is 68 % compared to that of the old custom code.

The significantly larger and more complex "Dice of Blood" game benefited more from size and complexity reduction in custom code than the "Flyswatter" game. The difference between size and complexity reduction in these two games can partly be explained by the amount of overhead added to the new custom code when using the framework, related to setting up data structures and implementing required interfaces. This overhead is more noticeable when framework is used in simpler games and will occupy a relatively larger portion of the code. Once the game gets more complex this overhead is less noticeable.

When comparing the old custom code and the new custom code it is clear that certain functionality was almost moved in its entirety to the framework, leaving only small sections of initialization code in the new custom code. Good examples are the Social Media functionality where the new custom code is 3 % of the old custom code for both the "Flyswatter" and "Dice of Blood" game, and the Motion Sensing functionality in the "Flyswatter" game where the new custom code is 29 % of the old custom code. The Game Logic custom code for the

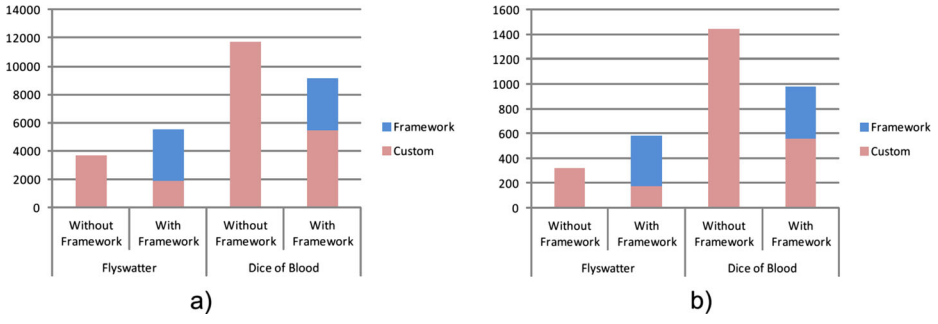


Fig. 11 **a** SLOC-P for "Flyswatter" and "Dice of Blood" games separated by custom and framework code **b** Cyclomatic complexity for "Flyswatter" and "Dice of Blood" games separated by custom and framework code

"Flyswatter" game was the only case where the new custom code was larger than the old custom code.

The "Fly Swatter" new custom code has a complexity density that is 79 % that of the complexity density in the framework components used. The "Dice of Blood" new custom code has a complexity density that is 89 % that of the complexity density in the framework components used. This indicates that the custom code contains less control flow logic statements per physical lines of code than the framework components. The games developed using the framework require less lines of code, which are also less complex. Defects that occur in the game are more likely to occur in the framework components than in the new custom code, but would only need to be resolved once.

6.2 Application responsiveness

Application responsiveness is of critical importance in game development. The proposed game development framework should by no means endanger the overall responsiveness of a gaming application. Acceptable overall response time, as stated in the most recent literature, is 2 s as the longest acceptable context recognition/context-switching delay, 0.5 s as the longest acceptable command response delay, and 0.2 s as the longest acceptable real time delay in gaming [4].

The evaluation experiment that was conducted includes measurement of the response time to the "Swat" gesture in the "FlySwatter" game, for the case (1) implemented using the proposed framework and (2) without the proposed framework. The obtained values are discussed with respect to the acceptable overall response time metrics stated before. Additionally, the response time is compared with the state-of-the-art HCI input medium—Kinect sensor, while detecting a "Punch" gesture which is the most similar in nature to the "Swat" gesture used in the "Flyswatter" game. The latency of a Kinect sensor detecting a "Punch" gesture is obtained from the work [26], which performed a detailed latency evaluation of a Kinect sensor while detecting gestures.

Figure 12 shows the response time to a "Swat" gesture, from the moment the gesture is intentionally started, until the moment the swatter is completely moved to a swat position on the TV screen, for 100 consecutive tests. The measurement is conducted by using a recording of the scene showing both the player and the TV screen. The first frame in which the player's

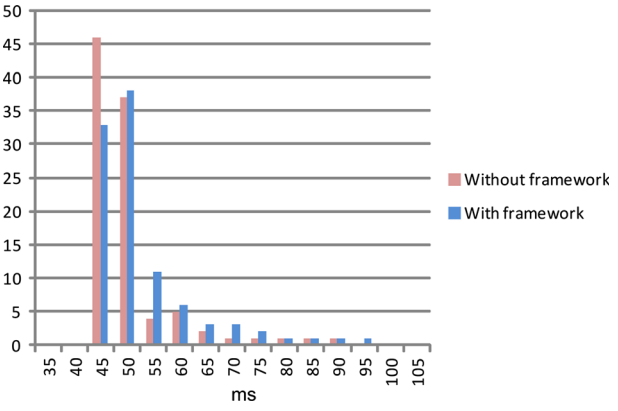


Fig. 12 Distribution graph of the responsiveness test results in ms for 100 consecutive "Swat" gestures in the "FlySwatter" game

hand moves back in the attempt to perform a "Swat" gesture, is cued as the measurement start time. The first frame, in which the swatter image on the screen is fully in the swatting position, is cued as the measurement end time. The experiment is conducted twice (first take for the application written without the proposed framework, and the second take for the application written with the proposed framework).

The results show that the usage of the proposed framework brings minimal overhead when compared with the implementation in which the framework is not used. The average response time when the framework is not used is 49.7 ms, whereas the average response time when the framework is used is 52.55 ms. These timings are well below all currently established thresholds for command responsiveness and entry-level gaming, indicating that the usage of the proposed framework is feasible in these scenarios.

The responsiveness of the Kinect sensor and the algorithms for "Punch" gesture detection in [26], which is the most similar gesture to the "Swat" gesture, is, on average approximately 980 ms. This is due to the nature of the sensor and video algorithms used for 3D segmentation and gesture classification. The main intention of the Kinect sensor is to allow game play without the need for additional controllers. Although in the TV-centric case the additional "controller", such as a smartphone, is required, we argue that these devices are ubiquitous and that people would increasingly carry one on their person at all times. Given the significantly better responsiveness when an accelerometer-based approach is used, enabled by the proposed framework, it is evident that TV-centric gaming concept surpasses the usability potential to that of "no controller" concepts.

7 Game experience evaluation and discussion

The TV-centric game experience evaluation was performed in two phases. In the first phase the TV-centric experience of playing turn-based games in both LAN and cloud scenarios was compared with the same games played in traditional tabletop version and online version using a desktop computer. In the second phase the experience of playing TV-centric real-time micro games overlaid on top of a TV program was evaluated. In both phases the evaluation was performed by collecting self-reported data using a questionnaire, supplemented by answers collected from open-ended interviews [36].

The evaluation was conducted over a period of two months, involving a total of 59 volunteers, 23 females and 36 males each of them taking part in the evaluation of both turn-based and real-time micro games. Their ages ranged between 25 and 40. All participants were familiar with classical card and board games, as well as a having at least casual gaming experience on desktop computers, gaming consoles or mobile devices. Each person involved in the evaluation watched at least two hours of TV daily.

The testing environment consisted of three rooms. Room 1 and 2 contained a TV connected to a set-top box, comfortable chairs and a table to simulate the atmosphere of a modern living room. Each of the players in those two rooms was supplied with a mobile device. The testing environment in room 1 is shown in Fig. 13. Room 3 was equipped with four desktop computers connected to the Internet and resembled an atmosphere of a working environment. Room 1 and 2 were used to play both the traditional versions of tabletop games and the TV-centric games. The purpose of having two rooms was to separate the two sets of players participating in the same session of a cloud version TV-centric turn-based game.

All mobile devices used had similar hardware characteristics, the same applied for the TVs, set-top boxes and desktop computers. The game performance was sufficient in terms of frame rate and latency ensuring smooth game play. The frame rate was kept at a minimum of 30 fps

Fig. 13 TV-centric testing environment

[10]. The latency between a player action on the mobile device and the result of that action displayed on the main screen had an upper limit of 100 ms for real-time micro games and 500 ms for turn-based games [9].

The turn-based game evaluation was performed separately for the "Crazy Eights" card game and the "Dice of Blood" board game by applying the same procedure to both games. The procedure consisted of evaluating the traditional, desktop, TV-centric LAN and TV-centric cloud version of the turn-based game by letting groups of 3 to 4 participants, which had previous social interaction, play each version in separate 15 min sessions, while being observed by a researcher. After each session the participants rated their agreement to the following game experience statement "I found the gaming experience to be enjoyable", in relation to the preceding session, on a Likert scale from 1 to 5 (1.strongly disagree, 2.disagree, 3.neutral, 4.agree, 5.strongly agree). After all four sessions were concluded, the entire group underwent an open-ended interview. The order of the sessions differed for each group.

The three real-time micro games "Egg-and-Spoon", "Flyswatter" and "Tomato Rating" were evaluated by letting a pair of participants, which had previous social interaction, play each game in separate 15 min sessions. During the game play a TV program was shown in the background using a documentary channel for the "Egg and Spoon" and "Flyswatter" micro games, and a movie channel for the "Tomato Rating", avoiding any commercial blocks. After each session the participants rated their agreement to the game experience statement, combined with the following ability to follow the TV program statement "I am able to retell another person what I saw on the television during game play", used to evaluate how well a person can simultaneously play the game and view the TV program. After all three sessions were concluded, the pair underwent an open-ended interview. The order of the sessions differed for each pair.

The game experience evaluation results for the turn-based games were gathered and analyzed by extracting the percentage of people who indicated agreement with the game experience statement by using the top two answers (4.agree and 5.strongly agree) for each of the four versions of both card and board game. The collected percentages are shown in Fig. 14. On the Y-axis the four versions are placed, on the X-axis the percentages of top two ratings for both card and board game are shown together.

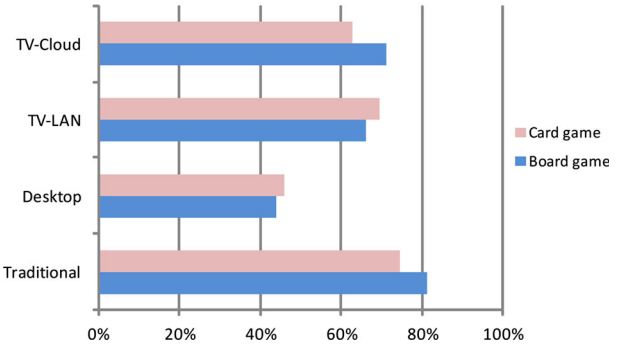


Fig. 14 Percentage of participants indicating agreement with game experience statement for turn-based games

The ratings indicate that the traditional versions of the tabletop card and board games were considered more enjoyable than their digital counterparts. However the TV-centric concept compares favorably to the version played on a desktop computer, showing that it comes close to the traditional version in terms of user experience. Comparison between TV-centric games showed that for the card game the LAN version was considered more enjoyable, while for the board game the cloud version was rated as enjoyable by more participants.

The following comments collected during the open ended interview were shared between different groups. Participants indicated they missed the physical elements of traditional game play such as shuffling a deck of cards and throwing real dice in the digital game versions, although the TV-centric version came closer to traditional versions by, for example, letting a player hold a hand of private cards. Something considered in favor of the digital versions was the use of animations, making the game world appear less static. Social interaction remained intact during the TV-centric sessions, but was lost comparing to the online version of the game using a desktop computer according to the comments. The participants enjoyed talking with each other during TV-centric game play and observing facial expressions of other players in the same room.

The game experience evaluation results for the real-time micro games were gathered and analyzed by extracting the percentage of people indicating agreement with the game experience statement by using the top two answers for each of the three micro games. The same procedure was performed using the ability to follow the TV program statement. The collected percentages are shown in Fig. 15. On the Y-axis the three micro games are placed, on the X-axis the percentages of the top two ratings for both statements are shown together.

A majority of the participants (between 69 and 79 %) indicated an enjoyable gaming experience. The “Tomato Rating” game was considered the most enjoyable and participants had the least amount of problems following the TV program, the latter can be explained because the game play element is very small and the point of the game is to focus on the TV program.

The “Egg-and-Spoon” game was rated the least enjoyable and the participants had the most difficulty following the TV program. Participants stated that for the “Egg-and-Spoon” game their attention was divided between the television screen and the mobile device, causing them to not pay attention to the TV program, also when looking at the television screen the player was focused on the bottom part of the screen where the game took place. A suggestion that was made by multiple participants was that they would rather play the “Egg-and-Spoon” game, while waiting for a TV program of interest to start.

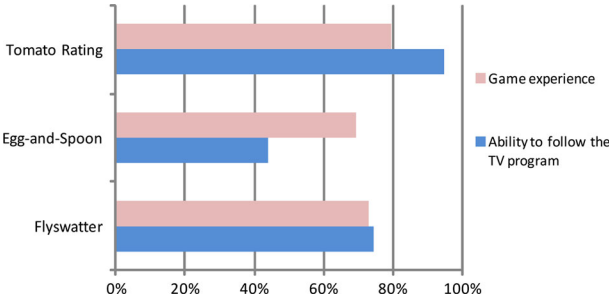


Fig. 15 Percentage of participants indicating agreement with game experience and ability to follow the TV program statements for real-time micro games

On a number of occasions while participants were playing the "Flyswatter" game they would treat the TV program background as part of the game, by incorporating objects seen on the TV program in the game play, by trying to swat it for example. Participants appreciated that the whole screen was used in the game and game graphics did not obstruct the actual television program.

8 Conclusion

This paper presented the innovative concept of TV-centric gaming, which involves an integration of mobile devices and DTVs connected to the Internet and already present in the living room to be used for recreating traditional tabletop games or creating novel games with access to both broadcast and broadband services. The concept intends to support social activities between family and friends playing a game while seated together in the same room or in different living rooms connected through cloud services. In order to greatly reduce time-to-market and cost for TV-centric gaming applications a development framework was created. Two turn-based games (one card and one board game) and three novel micro games that show their content overlaid on top of a television program were implemented using the framework and are discussed in detail showing the flexibility of the framework in the way its components are deployed to different locations and in the variety of game play supported. The framework and the developed applications are targeting a constantly growing market for Android multi-media consumer electronics. Both the framework and the games implemented using the framework were evaluated.

The framework evaluation was done by measuring framework effectiveness and application responsiveness. The effectiveness was measured by collecting the SLOC-P, cyclomatic complexity and complexity density metrics for two games, each of which was developed without and with the framework allowing comparison. The results show that the framework reduced both development and testing effort by moving complex logic required in a game implementation to re-usable components within the framework. Larger and more complex games will benefit more from the use of the framework compared to smaller games due to a decrease in relative overhead. Reduction in complexity applies the least to the custom game logic code that remains relatively complex when compared to custom code required to use other framework components, which has been in many cases reduced to initialization code containing less control flow logic. However developers benefitted from the guidance that the framework

provided in structuring the game logic. The use of the framework in general resulted in a higher degree of separation of responsibilities in each implemented class in the new custom code when comparing with the old. Using the framework forced developers not to mix code within a single class. The application responsiveness was measured for a "Swat" gesture on a game implemented without and with the framework, additionally it was compared with the responsiveness measurements for a "Punch" gesture using the Kinect sensor. The responsiveness measurements collected from the game developed without and with the framework show that although the usage of the framework introduced an overhead causing a slightly larger response time, the response times for both versions were well within acceptable performance limits and compared favorably with response times of a system using a more advanced method of gesture recognition based on the Kinect sensor.

The TV-centric gaming experience was measured by collecting and analyzing self-reported data using a questionnaire combined with additional observations from 59 volunteers. The questions asked for turn-based games were used to compare the experience while playing the same games in four different scenarios: traditional, desktop, TV-centric LAN and TV-centric cloud version. The questions asked for real-time micro games were used to compare the different micro games among themselves and to analyze how well participants can simultaneously play the game and follow the TV program running in the background. Several key observations were made that could be used as guide lines for the development of TV-centric games. Traditional board and card games have a social appeal that cannot be easily replaced in digital form, the TV-centric approach comes close by preserving face-to-face social contact and forms of tangible interaction with the game (for example: holding the mobile device showing a hand of cards). For more complicated strategy board games with a team element it can be beneficial for members of opposite teams to be placed in different rooms (playing the cloud version of the game) allowing team members to discuss strategy among themselves. The casual nature of the micro games and their use of intuitive motion sensor control combined with the integration of broadcast functionality can enrich the TV viewing experience. Simpler games perform better in terms of game experience and TV viewing ability on the condition that care is taken to ensure that the game graphics overlaid on top of television content are minimal and non-intrusive. When applicable it is preferred that a player interacts directly with objects in the game world through touch on the second screen supplemented by motion sensing and haptic feedback, rather than rendering explicit buttons on the second screen that can be touched to control the game. Dividing the attention of a player between two screens during action games such as the real-time micro games prevents players from immersing themselves in game play, as well as limit the player's ability to follow the TV program, however for turn-based games which by their nature are slower paced, the use of separate screens for private and public information improves the game play. To efficiently utilize the space available on a large public display such as a TV, the dimensions of the game world typically shown in full on a public display should have a similar aspect ratio as commonly used on televisions, such as 16:9. Mobile devices have a large degree of variation in aspect ratio and a limited amount of screen space which makes them more suited to show a partial view of the game world or show a separate view supplemental to the public view.

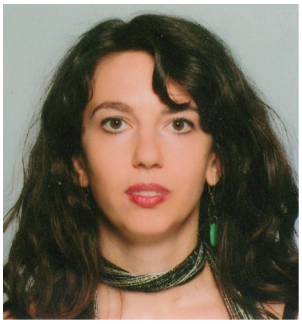
Future work could include a non-Java implementation of browser-based clients that would connect to a web service in the cloud. This would enable the development of platform independent games; although a potential trade-off would be the loss of motion sensing abilities in the game client. A possible extension to integrating broadband content into game play would be to investigate methods of triggering events in a game world based on television content.

Acknowledgments This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, project No. 32041, and 44009.

References

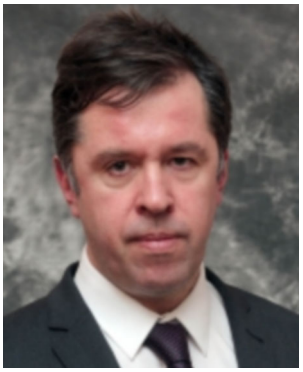
1. Baek J, Yun B (2008) A sequence-action recognition applying state machine for user interface. IEEE Trans Consum Electron 54(2):719–726. doi:10.1109/TCE.2008.4560153
2. Barkhuus L, Brown B (2009) Unpacking the television. ACM Trans Comput Hum Interact 16(3):1–22. doi:10.1145/1592440.1592444
3. Bhandari S, Lim YK (2008) Exploring gestural mode of interaction with mobile phones. In: CHI'08 extended abstracts on human factors in computing systems, pp 2979–2984. doi:10.1145/1358628.1358794
4. Bjelica MZ (2013) Methods of implementation of context-aware platforms and context-aware user interfaces for applications in consumer electronics. Doctoral dissertation, Faculty of Technical Sciences, University of Novi Sad, Serbia, pp 1-361
5. Bjelica MZ, Zdravkovic V, Punt M, Teslic N (2013) TV-centric gaming applications for Android OS: architecture and a framework. In: The IEEE international conference on consumer electronics (ICCE '13), pp 667–668, January 2013. doi:10.1109/ICCE.2013.6487064
6. Boehm B, Abts C, Chulani S (2000) Software development cost estimation approaches—a survey. Ann Softw Eng 10(1–4):177–205. doi:10.1023/A:1018991717352
7. Boertjes E (2007) ConnecTV: share the experience. In: EuroITV 2007, pp 139–140. doi:10.4018/978-1-60566-656-3
8. Chuang YL, Liao CW, Chen WS, Chang WT, Cheng SH, Zeng YC, Chan KH (2013) Use second screen to enhance TV viewing experiences. Cross-cultural design. Methods Pract Case Stud 8023:366–374. doi:10.1007/978-3-642-39143-9_41
9. Claypool M, Claypool K (2006) Latency and player actions in online games. Commun ACM 49(11):40–45. doi:10.1145/1167838.1167860
10. Claypool M, Claypool K, Damaa F (2006) The effects of frame rate and resolution on users playing first person shooter games. In: The ACM/SPIE Multimedia Computing and Networking (MMCN'06) Conference, San Jose, CA, USA, pp 1–11. doi:10.1117/12.648609
11. Coppens T, Trappeniers L, Godon M (2004) AmigoTV: towards a social TV experience. In: Second European conference on interactive television: enhancing the experience, Brighton, pp 159–162
12. Courtois C, D'heer E (2012) Second screen applications and tablet users: constellation, awareness, experience, and interest. In: The 10th European conference on Interactive tv and video, pp 153–156. doi:10.1145/2325616.2325646
13. Cruickshank L, Tsekleves E, Whitham R, Hill A, Kondo K (2007) Making interactive TV easier to use: interface design for a second screen approach. Des J 10(3):41–53. doi:10.2752/146069207789271920
14. Döring T, Shirazi A. S, Schmidt A (2010) Exploring gesture-based interaction techniques in multi-display environments with mobile phones and a multi-touch table. In: The international conference on advanced visual interfaces (AVI'10), pp 419–419. doi:10.1145/1842993.1843097
15. Fallahkhair S, Pemberton L, Griffiths R (2007) Development of a cross-platform ubiquitous language learning service via mobile phone and interactive television. J Comput Assist Learn 23(4):312–325. doi:10.1111/j.1365-2729.2007.00236.x
16. Gauntlett D, Hill A (1999) TV Living: television, culture and everyday life. Routledge, London
17. Gill GK, Kemerer CF (1991) Cyclomatic complexity density and software maintenance productivity. IEEE Trans Softw Eng 17(12):1284–1288. doi:10.1109/32.106988
18. Häsel M (2011) Opensocial: an enabler for social applications on the web. Commun ACM 54(1):139–144. doi:10.1145/1866739.1866765

19. Itterheim S, Löw A, Chew B, Hillerson T (2012) Learn Cocos2D 2. Apress, Berkeley
20. Joselli M, Clua E (2009) Grmobile: a framework for touch and accelerometer gesture recognition for mobile games. In: VIII Brazilian symposium on games and digital entertainment (SBGAMES), pp 141–150. doi:10.1109/SBGAMES.2009.24
21. Kemp R, Palmer N, Kielmann T, Bal H (2012) Cuckoo: a computation offloading framework for smartphones. Mob Comput Appl Serv 76:59–79. doi:10.1007/978-3-642-29336-8_4
22. libGDX (2013) https://github.com/libgdx/libgdx/. Accessed 12 February 2014
23. LocMetrics (2006) http://www.locmetrics.com. Accessed 5 Sept 2013
24. Luyten K, Thys K, Huypens S, Coninx K (2006) Telebuddies on the move: social stitching to enhance the networked gaming experience. In: 5th ACM SIGCOMM workshop on Network and system support for games, no 18. doi:10.1145/1230040.1230084
25. Magerkurth C, Memisoglu M, Engelke T, Streitz N (2004) Towards the next generation of tabletop gaming experiences. In: The graphics interface conference (GI'04), pp 73–80
26. Masood SZ, Ellis C, Nagaraja A, Tappen M. F, LaViola J. J, Sukthankar R (2011) Measuring and reducing observational latency when recognizing actions. In: 2011 I.E. International Conference on Computer Vision Workshops (ICCV Workshops), pp 422–429. doi:10.1109/ICCVW.2011.6130272
27. McAdam C, Brewster S (2011) Using mobile phones to interact with tabletop computers. In: The ACM international conference on interactive tabletops and surfaces, pp 232–241. doi:10.1145/2076354.2076395
28. McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320. doi:10.1109/TSE.1976.233837
29. Metcalf C, Harboe G, Tullio J, Massey N, Romano G, Huang E. M, Bentley F (2008) Examining presence and lightweight messaging in a social television experience. ACM transactions on multimedia computing, communications, and applications 4(4):no 27. doi:10.1145/1412196.1412200
30. Nathan M, Harrison C, Yarosh S, Terveen L, Stead L, Amento B (2008) CollaboraTV: making television viewing social again. In: 1st international conference on designing interactive user experiences for TV and video, pp 85–94. doi:10.1145/1453805.1453824
31. Nguyen V, Deeds-Rubin S, Tan T, Boehm B (2007) A SLOC counting standard. In: The 22nd international annual forum on COCOMO and systems/software cost modeling, Los Angeles, USA, October 2007
32. Nielsen Company (2011) http://www.nielsen.com/us/en/reports/2011/cross-platform-is-the-new-norm.html. Accessed 9 Feb 2014
33. Römer K, Domnitcheva S (2002) Smart playing cards: a ubiquitous computing game. Pers Ubiquit Comput 6(5–6):371–377. doi:10.1007/s007790200042
34. Scheible J, Ojala T, Coulton P (2008) MobiToss: a novel gesture based interface for creating and sharing mobile multimedia art on large public displays. In: The 16th ACM international conference on Multimedia, pp 957–960. doi:10.1145/1459359.1459532
35. Schroeder J, Broyles B (2013) AndEngine for Android game development cookbook. Packt Publishing Ltd., Birmingham
36. Tullis T, Albert W (2010) Measuring the user experience: collecting, analyzing, and presenting usability metrics. Morgan Kaufmann, Burlington
37. Vajk T, Coulton P, Bamford W, Edwards R (2008) Using a mobile phone as a 'Wii-like' controller for playing games on a large public display. Int J Comput Game Technol 2008:1–6. doi:10.1155/2008/539078
38. Vidakovic M, Maruna T, Teslic N, Mihic V (2012) A java API interface for the integration of DTV services in embedded multimedia devices. IEEE Trans Consum Electron 58(3):1063–1069. doi:10.1109/TCE.2012.6311357
39. Wang SC, Chung TC, Yan KQ (2011) A new territory of multi-user variable remote control for interactive TV. Multimedia Tools and Appl 51(3):1013–1034. doi:10.1007/s11042-009-0435-0
40. Williams D, Ursu MF, Meenowa J, Cesar P, Kegel I, Bergström K (2011) Video mediated social interaction between groups: system requirements and technology challenges. Telematics Inform 28(4):251–270. doi:10.1016/j.tele.2010.11.001
41. Zamith M, Montenegro A, Clua E, Joselli M, Feijo B, Leal R, Valente L (2011) A distributed architecture for mobile digital games based on cloud computing. In: Brazilian symposium on computer games and digital entertainment (SBGames'11), pp 79–88, November 2011. doi:10.1109/SBGAMES.2011.13
42. Zechner M, Green R (2011) Beginning Android 4 games development. Apress Media, Berkeley

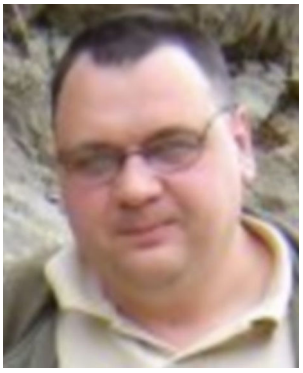


Marija Punt received the B.Sc. and M.Sc. degrees in electrical engineering from the University of Belgrade, Serbia. She is currently a teaching assistant with the department of Computer Engineering, School of Electrical Engineering, University of Belgrade and she is also working towards her Ph.D. degree at the department of Computer Engineering. Her research interests include computer architecture, digital systems simulation, and consumer electronics.

Milan Z. Bjelica has received B.Sc., M.Sc. and Ph.D. degrees from the Faculty of Technical Sciences, University of Novi Sad, Serbia. He is currently employed as a software design engineer at RT-RK Institute for computer based systems, and as the teaching assistant at the University of Novi Sad. His research interests and expertise include novel immersive context-aware systems, usability assessment and home automation systems, frequently applied to the different consumer electronics devices.



Vladan Zdravkovic received his B.Sc. and M.Sc. in electrical engineering from the University of Belgrade, Serbia. He received the B.Sc., M. Sc. and Ph.D. degrees from the University of Arts in Belgrade, Serbia. He is affiliated to the Sheffield Hallam University in England. His research interests include computer animation and 3D modeling.



Nikola Teslic received the B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the Faculty of Technical Sciences, University of Novi Sad, in 1995, 1997 and 1999 respectively. He is the professor at the department for Computer Engineering and Computer Communication, University of Novi Sad. His research interest covers design of real time systems, audio and video processing and the design of testing systems.