– GAMEWORK – A CUSTOMIZABLE FRAMEWORK FOR PERVASIVE GAMES

Christoph Stach
Universität Stuttgart
IPVS
Universitätsstraße 38
70569 Stuttgart
christoph.stach@ipvs.uni-stuttgart.de

ABSTRACT

The number of pervasive games is growing in which players have to interact with their environment in order to control their avatar. While in the past they required immense hardware equipment, nowadays, smartphones have all required functionality built-in already. In addition, there is an upcoming trend towards software, supported with new content and knowledge by an active community. By combining these two trends, a new genre of computer games arises, targeting not only established gamers but also new audiences.

We build four customization classes differing in required player's programming knowledge, support by the games and scale of adaption potential. With *Gamework* we present a framework for pervasive games on modern smartphones simplifying user-driven game customization for the identified customization methods in this paper. We also present two games implemented with this framework. Concluding, we show the potential of these games as well as the potential of our framework. We also report on early experiences in exploiting the customization approach of our framework.

KEYWORDS

Pervasive, mobile, customizable, games, framework, smartphones.

1. INTRODUCTION

Although pervasive games may differ in many aspects, they all somehow map an extract of real world's context on a virtual game [Magerkurth et al. (2005)]. Generally, this happens through a GPS-based sensor which tracks a player's position and uses it to control the avatar in the game. Therefore, most common pervasive games use massive and expensive hardware assignments [Broll et al. (2006)]. Albeit, there are approaches such as *AR-Soccer* [Paelke et al. (2004)], a penalty game with a virtual ball and goal, or *Capture the Flag* [Cheok et al. (2006)], where two teams have to defend a flag while capturing the opponent's one, almost all well-known pervasive games focus on augmented reality systems – computer-based systems stimulating any human sense especially vision [Azuma (1997)] – including backpacked laptops or even head-mounted displays [Wetzel et al. (2008)]. In *Human Pacman*, a pervasive version of the original *Pacman* game, a player controls either the Pacman or a ghost by moving through the streets of Singapore and has to collect points or catch the Pacman in order to win. Therefore, every player needs a special vest or a backpack equipped with nearly all components of a laptop [Cheok et al. (2004)]. Although, the *REXplorer*, an educational city-game provided by the Regensburg Experience museum, does not need extensive technical equipment, it still has to provide due to special technological features all players with all devices that are used in the game, which limits the number of simultaneous players [Ballagas et al. (2007)]. Even though, these games are a lot of fun, the effort needed to set up a game session is too high to arrange them regularly.

On the other hand, modern smartphones are shipped with components such as a big touchscreen, a GPS sensor, Bluetooth, Wi-Fi, and much more. Thus, all of these devices fulfill nearly all requirements set by pervasive games. In face of the given conditions, still the number of pervasive games for smartphones is very limited. Furthermore it is astonishing that none of these few existing games allows players to bring in their own ideas and alter or enrich the original game idea, though modern operating systems for mobiles enable everybody to develop their own additional applications for those devices [Hall and Anderson (2009)]. Without such a feature games might lose their attraction very rapidly [Eirund et al. (2004)]. In particular, pervasive games qualify for small customizations, as placing the playing field to another (real world) location, adding new tasks up to extending the game logic [Wolff and Grüter (2008)]. We conclude that the programming standards for such applications are too high for an average player.

Hence, we initially identified different customization techniques. Afterwards, we analyzed what kind of game adaption each of them could be used for, classified our audience depending on their knowledge in programming and determined how each of them could be assisted by a framework. Based on this preliminary work, we created a framework for pervasive games on mobile devices in order to enable a shortened developmental period and to involve all players in the development process by giving them the tools to customize a game to a certain degree. While some frameworks for pervasive games already exist (e.g. *FRAP* [Tutzschke and Zukunft (2009)] or our former approach for mobile browser games [Brodt and Stach (2009)]), our new approach supports customizable pervasive games which is a unique feature. The scale a game can be customized with our approach is determined by the game's nature as well as the player's expertise. But, even players with limited programming experience will be able to make small changes on existing games within no time, while programming experts are supported in developing new games or even new game genres. To demonstrate this, we present prototypes of two pervasive games implemented with our framework and use them to host a workshop in order to evaluate *Gamework*. As far as possible we kept our target architecture open to stay independent from any unpredictable trend affecting smartphones' technology.

The remainder of this paper is structured as follows: Section 2 discusses different existing customization techniques and categorizes them. Section 3 describes the framework itself while Section 4 presents two different games implemented with the framework. The outcome of our evaluation is presented in Section 5. Finally, Section 6 discloses future work while Section 7 concludes the whole issue.

2. DEGREES OF CUSTOMIZATION

In some preliminary work, we studied game's nature and how players can influence it. As a result of this work, we were able to distinguish four different classes of customization techniques in games. These classes vary in effort as well as scope of supply and services. In the following subsections, we deal with game customization by using context data or user-generated content in games and adapting the game-flow or even creating new games by only reusing the framework's artifacts. Because of focusing on pervasive games against (human) players we will not deal with dynamic (AI) adaption techniques, as presented in [Hunicke (2005)]. We will conclude this section by a short comparison of these techniques.

2.1 Context in games

The easiest way to customize a game is putting the game-field right where players are located at the moment. This adaption technique does not require any programming skills from a player. Nevertheless, it will give a player an individual game experience because no other plays exactly that game. On the other hand, game designers will have to include this customization ability in their game logic and not every type of game is appropriate handling with dynamic game-field positions. E.g., the placement of games in which the players have to interact heavily with their environment should be chosen wisely, because natural obstacles can hamper accomplishing an aim unnecessary or even inhibit it completely. Therefore, such kind of games must go without the presence of fixed levels. [Schlieder et al. (2006)] reckon that especially location-based versions of classical board games suits to be modified this way.

SYGo, which is actually a pervasive adoption of the famous German board game Scotland Yard by Ravensburger, represents such a kind of game [Schmatz et al. (2009)]. The game-field is always virtually set in the center of London, but in reality, the player's current position specifies where the game actually takes place. This kind of game is characterized by ignoring a player's absolute position but usually analyzing only the movement relative to the starting position. While *SYGo* automatically adapts the game-field's position it still forces all players to gather within a certain region close to its center. We will even present a game without this restriction in Subsection 4.2.

As a matter of course, this kind of customization has to be provided by each game itself and not by a framework. But, we tried to support those players who want to use such a feature in their games in the game development process by adding geo functions to our framework, e.g. for calculating the relative distance to a given location.

2.2 User-generated content

Another way allowing players to customize a game is to give them the possibility to add new content to an existing game. While this is nothing new in the Web 2.0 environment, it's rarely used in games. We are convinced that especially location-based games are suited for this kind of customization. [Krumm et al. (2008)] defines pervasive user-generated content as the possibility to add some new information to a community-based system, e.g. some comments to given points of interest. Therefore, it requires many voluntary contributors. On the other hand, the binding within the community is tightened, thereby.

E.g., user-generated content could be used to add a new target which actually implies creating a new level or to ease an existing search by adding some hints in a geocaching-like game. Our framework provides functions to store additional information such as pictures or some text to existing objects and allows the player to create completely new points of interest. In addition, *Gamework* includes a map-editor with a GUI which allows even a non-technical player to create, modify or delete any object in the game just by clicking on a map. Nevertheless, this feature must be supported by the game. If a game manages its objects by itself, the *Gamework* editor cannot interact with them.

2.3 Game-flow adaption

To customize games in a major way, its game-flow has to be modified. [Taylor et al. (2006)] introduces a technique to visualize a flow-graph for every level in an UML-like style. In the graph, not only the game-flow with all alternatives to solve a level is represented but also the storyline and any scripted event. By merging all flow-graphs a single graph is generated which represents the whole game. Then, all there is left to do in order to create a new similar game is to modify or rearrange the graph's nodes or edges.

The easiest customization using this technique would be arranging the levels in a new order or removing some of them by moving or deleting some edges. Provided with some examples, an interested player should also be able to add new states or branches to the graph, after a while. We considered that technique already in our framework design phase by representing every game as a finite state machine. So, a player who understands Java code is able to modify the game-flow, in a simple way. For players, inexperienced in the field of programming, a WYSIWYG editor would be auxiliary but is not a part of *Gamework* at this moment.

2.4 Creation of new games from scratch

While there are many parameters making a game unique e.g. story, look or game-flow, some components in games are used by nearly all games independent from their genre (e.g. the use of a finite state machine for its logic or the need to map an item to a field) [Rollings and Adams (2003)]. So, we tried to gather those common components in the modules of our framework to allow designers to easily reuse them. In our case, these modules include an extensible object type for game items, single- and multi-players a login system or a graphical editor for connecting virtual objects to coordinates or manipulating them, just to name a few. In a framework for non-pervasive games, these packages would probably look slightly different. Due to *Gamework's* strictly modular structure, designers will only have to create the unique parts of the game while they can include the common ones from our framework's packages.

2.5 Comparison

To conclude this section, we summarize the prominent features of the four described customization techniques in Table 1. As shown, the degree of customization possibilities rises with the experience of the player while the required support by each game decreases. Anyhow, every kind of player can benefit from *Gamework*.

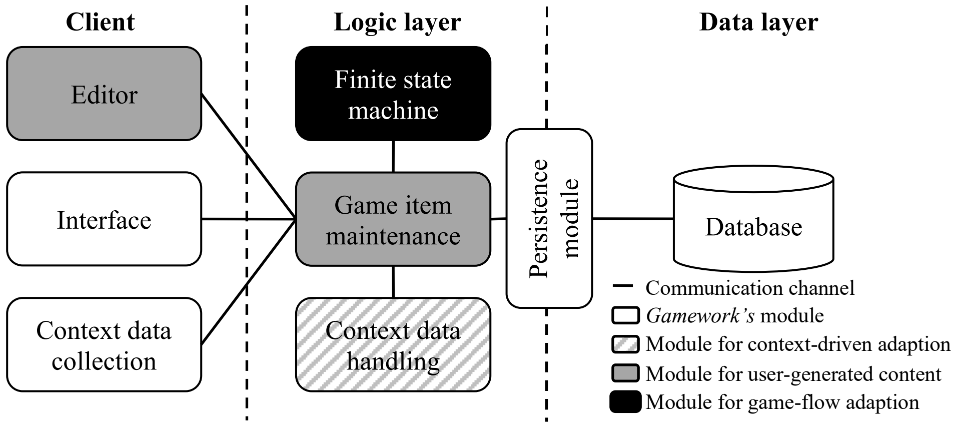
Table 1. Review of the four customization techniques

Game aspect	Context in games	User-generated content	Game-flow adaption	Creation of new games from scratch
Framework support	Geo functions (functions for distance calculation)	Graphical editor (alter characteristics of game items)	FSM (add new states or change game-flow)	Modular structure (reuse of common modules)
Required game support	Very high	High	Moderate	Non
Programming skills required	Non	Non	Moderate	Very high
Degree of customization	Low	Moderate	High	Unlimited

3. FRAMEWORK

This approach gears on our former research ([Brodt and Stach (2009)]) using a classical three tier architecture. As shown in Figure 1 *Gamework's* main modules rest on a central server, while the mobile device is simply used as a thin client. This means that in addition to the game's presentation the context data retrieval is merely accomplished by it. Therefore, not only the hardware requirements towards the mobile device is held imaginable slight but also the required adjustments caused by a change in target architecture's technology would be kept to a minimum.

Figure 1. *Gamework's* architecture



The framework's central component is formed by the generic game item maintenance module. New objects (e.g. players, items, etc.) can easily be modeled by listing its characteristics such as name, value, measurements, and so on. This module allows the editor to embed user-generated content to a game. It also forwards the collected context data (or input data) to the context data handling component which provides functions as calculating relative position data from absolute position data or setting the game field to the player's current position. While the mobile device submits only absolute positioning data to the logic layer, we developed a context data handling module which maps these coordinates to an in-game location or calculate a player's relative movement from these data in order to enable context-driven adaption. On top of the game item maintenance module, a finite state machine is available to model all game states (for instance login → start → … → end) as well as its different stages and even every task within these stages. Any modification here has an impact on the game-flow. Finally, our persistence module can transfer all relevant game data to the data layer in order to store them in or collect them from the utilized database allowing load / save functionality in the games. Since all of these modules work independently from each other and provide well-documented interfaces, they also can be separately used from the rest of the framework. So, completely new games can be implemented much faster by reusing any frequently recurring functions wrapped in these modules.

For our first implementation, we choose Android-bases mobile devices. We did so because of two reasons: First of all, we are convinced that Java is one of the most popular languages by now (according to [DedaSys LLC (2009)]). Furthermore, [Kang (2009)] identifies these devices as the fastest-growing smartphone group in 2009 by far. Even though, many developers are prejudiced against the use of Java for fast applications such as games, new benchmarks prove, that these preconceptions against Java are either obsolete or just false [Ingles (2006)]. Furthermore, there is a growing number of successful serious games based on Java [Davison (2005)]. The game logic modules run on a Tomcat server. Communication with the backend is arranged by Hibernate while Apache CXF does this with the frontend. For the administration interface Grails proved to be the best solution. For storage issues, we used a H2 database. But the kind of database as well as the frontend is easily replaceable.

4. PROTOTYPES

In order to demonstrate *Gamework's* capability, we implemented two small different pervasive games: *1-2-3…* and *P^2 – Pervasive Pairs*. *1-2-3…* is a very simple conversion of a geocaching-like search game where virtual items are placed on a map and the player has to collect them. *P^2* is a location-based version of the well-known tabletop game *Memory*® by Ravensburger.

4.1 *1-2-3…*

To give a player a small showcase how to use the GPS-interface and our finite state machine, we developed *1-2-3…*. In the beginning a player creates a course by placing an arbitrary number of targets all over an area and assigns them an order in which they have to be collected. Then he invites a friend to test his new track. The friend sees the corresponding maps detail, her current position and the targets position. With this information, she has to find the best way connecting all set points. Then, she has to reach all these targets in the given order as fast as possible. Whenever her position corresponds with a target's position, the game logic checks whether she has already achieved all prior goals and activates the next goal.

1-2-3… can be customized by creating a new or expanding an existing track. Each target is represented by a new state in the finite state machine. To place a new aim, a player has to create a new instance of a state object. However, he is given not only the possibility to adjust or set a new position but also to add some conditions to each state such as small tasks which has to be met before the next state can be reached. More complex than adding new states is changing the order of the states because many transitions have to be edited by hand. In a new version, we plan to describe the states and transitions with a more readable description language in an external file to simplify the creation and modification of a course.

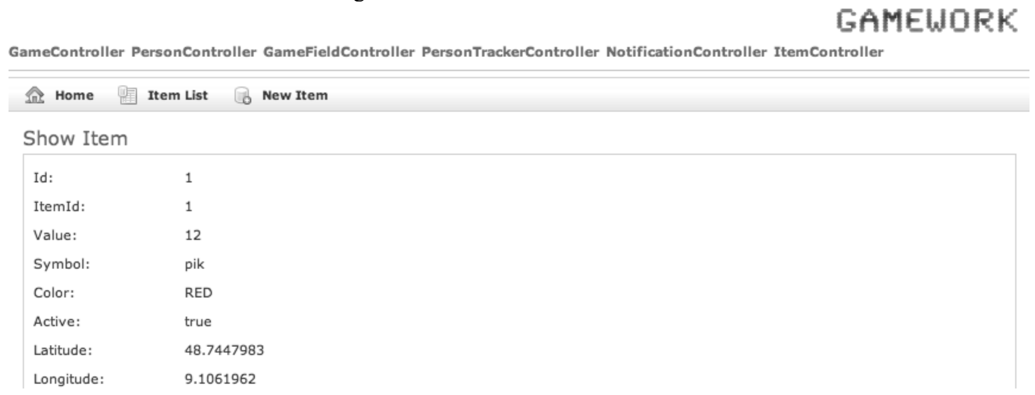
4.2 *P^2 – Pervasive Pairs*

In *Pairs*-games at least two players have to collect as many game cards as possible until no more cards are left. The player with the most cards is the winner. To collect a card, each player in turn is allowed to look at two cards lying covered on a table. If the two cards show the same picture the player may collect this set.

The *P^2* version adopts the basic game idea and adds some new aspects to make it much more dynamic and appropriate to the pervasive setting. Firstly, the cards are no longer real-world items but virtual objects. So, neither their amount nor their images and not even their sizes are predetermined. They are randomly positioned in the player's surroundings. Consider that this means it is not necessarily required to have all players at the same location! When a player reaches the position of a card's center (within a certain tolerance that compensates for GPS inaccuracies or the presence of real world obstacles), its image is revealed only to the corresponding player and remains locked to all other players until a second card is selected. If the motives match the card is taken out of the game and the player is rewarded. A positive match could be the same image, but also any predefined pattern, such as the right combination of a celebrities' given- and surname. Maybe, it is dubious by now why the first card has to be locked for all other players. This results from the other big difference between the tabletop game and this pervasive version. In *P^2* all players may move at the same time and will not have to wait until their opponents have finished their turns. We assumed that this modification is necessary to boost the game's entertainment factor – without this alternation all players would have to stay at their current position for most of the time. So not only memorize abilities determine the game's outcome, but also the player's fitness level.

For the game's implementation, we used our framework presented in Section 3. As a consequence, not only the development time was dramatically shortened but also the whole game is customizable to a high degree. For example, the maximum number of players, the number of cards, the size of the game field, the position of the game field, the cards motive, the pairs value, etc. can be adapted to any desired value by only changing one constant. With a bigger effort but still very simple coding we have experimented with forced player-dependent waiting times between the uncovering of two cards for balancing reasons.

Figure 2. *Gamework's* editor interface


GAMEWORK
GameController PersonController GameFieldController PersonTrackerController NotificationController ItemController
Home
Item List
New Item

Show Item

Id:	1
ItemId:	1
Value:	12
Symbol:	pik
Color:	RED
Active:	true
Latitude:	48.7447983
Longitude:	9.1061962

Figure 2 shows the *Gamework* editor interface used for customizing the cards in P^2. In addition to common attributes for every element, e.g. Id, Latitude or Longitude, game specific entries as in this case 'Symbol' or 'Color' can also be managed by the editor. Within a few clicks, new cards can be added to the game or existing cards can be modified.

4.3 Comparison

Table 2 summarizes the fundamental game aspects of *1-2-3…* and P^2 and documents that even our two very simple prototypes differ so much, that they can be considered as two exponents for different game genres.

Table 2. Comparison between *1-2-3…* and P^2 concerning different game aspects

Game aspect	*1-2-3…*	P^2 – *Pervasive Pairs*
Number of players	Single- and Multiplayer	Multiplayer
Multiplayer mode	Asynchronous	Synchronous
Setting	Real world	Augmented world
Game type	Geocaching	Tabletop game
Use of position data	Absolute coordinates	Relative

5. EVALUATION

After we had completed our framework and our two prototypes, we decided to test and evaluate the usability not just by ourselves or experienced students, but also by pupils who have just started to learn some programming techniques at school. Therefore, we organized a three-day workshop for 12 pupils in which we gave a brief overview of the *Gamework* framework and provided them with a couple of Android-Smartphones. Then, they were allowed to spend one day on their own, not only by playing *1-2-3…* and P^2, but also by customizing these two games and trying to create their own small games without any guidance. That way, we wanted to evaluate how self-explanatory our framework is. After the workshop, we hand out evaluation sheets to find out how they appraised their programming knowledge, what experiences they gained, what functionality they missed and whether the framework met their expectations. Due to a lack of sufficient hardware and participants this evaluation has to be seen as a first attempt to determine *Gamework's* benefit.

Their feedback made one thing very clear: *Gamework* is actually able to provide even an inexperienced player with everything needed to make small customizations on existing games or actually create new content e.g. new levels. For more complex changes, neither the time nor their experience was sufficient. The only slightly negative statements tended towards the unmet wish for chances to create a game with brilliant 3D graphics within an afternoon without touching a single line of code. All participants stated that they really liked the possibilities of *Gamework* and enjoyed the completely new experience of pervasive games.

6. FUTURE WORK

We see our framework as a huge contribution towards customizable pervasive games' implementation, especially location-based ones. In the future, we are planning to implement other methods to monitor the player's position, such as Cell-ID or locating via Wi-Fi whenever GPS is not available. This makes games in areas with much atmospheric interference possible and even indoor games would be supported. Another issue concerns the very limited battery life and the huge energy consumption caused by the GPS.

Even though, small customizations can be made easily by changing only some constants' value, we plan to source these characteristics completely out to an XML configuration file and support the import of such a file directly by the framework. So, not only the readability can be increased but also the customizability for unskilled players because they merely have to handle a common text editor and no recompilation is required after these changes.

While those plans only affect the improvement of functions already implemented in the framework, we also think about the framework's enhancement heading towards the support of *'real'* pervasive games. There are so many interesting sensors built in modern smartphones (and newer models will surely come up with even more) which we are not supporting at the moment. This could involve games where the camera is used to shoot photos of a certain location or record some typically sound with the microphone to prove that a player was actually at the specified place. This type of game would be appropriate to enable user-generated content as new stages in different regions or expand existing ones by adding new targets to shoot or record.

7. CONCLUSION

In this paper, we presented a classification method for customization techniques for pervasive games. We characterized each of these classes regarding to their adaption potential, the player's programming expertise and to which degree a framework can support them. With this preliminary work, we implemented a framework supporting each of the identified customization techniques. To prove its advantages, we used it to implement two pervasive game prototypes. Our main focus always rested on generating games which are easily adaptable on different degrees according to the player's knowledge. Since even players with limited programming experience should be able to profit by our framework, we evaluated its usability by a workshop for beginners in computer science.

We assume that the market for common computer games will stagnate or even reduce while the number of pervasive games, especially on mobile devices, will rise in the future due to better and cheaper hardware. Communities will form, providing these games with new content comparable with twitter or Facebook today. Henceforward, teenagers who will probably form the majority within these Communities will be able to get in touch with programming duties in a playful manner due to *Gamework*.

ACKNOWLEDGEMENT

We would like to thank our colleagues Andreas Brodt and Marko Vrhovnik as well as our students Florian Berg, Patrick Hilsbos and Florian Mack for supporting our work. Furthermore, we would like to thank Dr. Wieland Holfelder, Google Germany, for equipping us with Android-Smartphones, needed for the framework's development and prototypes' testing.

REFERENCES

[Azuma (1997)] Azuma, R. T.: 1997, A Survey of Augmented Reality, *Presence: Teleoperators and Virtual Environments* **6**(4), 355–385.

[Ballagas et al. (2007)] Ballagas, R. A., Kratz, S. G., Borchers, J., Yu, E., Walz, S. P., Fuhr, C. O., Hovestadt, L. and Tann, M.: 2007, REXplorer: A Mobile, Pervasive Spell-casting Game for Tourists, *CHI '07 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '07, pp. 1929–1934.

[Brodt and Stach (2009)] Brodt, A. and Stach, C.: 2009, Mobile ortsbasierte Browserspiele, *INFORMATIK 2009: Tagungsband der 39. GI-Jahrestagung, 28.9. – 2.10.2009, Lübeck*, Vol. 154 of *LNI*, pp. 1902–1913.

[Broll et al. (2006)] Broll, W., Ohlenburg, J., Lindt, I., Herbst, I. and Braun, A.-K.: 2006, Meeting Technology Challenges of Pervasive Augmented Reality Games, *Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '06, pp. 28:1–28:12.

[Cheok et al. (2004)] Cheok, A. D., Goh, K. H., Liu, W., Farbiz, F., Fong, S. W., Teo, S. L., Li, Y. and Yang, X.: 2004, Human Pacman: A Mobile, Wide-area Entertainment System Based on Physical, Social, and Ubiquitous Computing, *Personal and Ubiquitous Computing* **8**(2), 71–81.

[Cheok et al. (2006)] Cheok, A. D., Sreekumar, A., Lei, C. and Thang, L. N.: 2006, Capture the Flag: Mixed-Reality Social Gaming with Smart Phones, *IEEE Pervasive Computing* **5**(2), 62–69.

[Davison (2005)] Davison, A.: 2005, *Killer Game Programming in Java*, O'Reilly Media, Inc.

[DedaSys LLC (2009)] DedaSys LLC: 2009, Programming Language Popularity, http://www.langpop.com.

[Eirund et al. (2004)] Eirund, H., Grüter, B. and Mielke, A.: 2004, Der Spieler macht das Spiel – Mechanismen der Autorenrolle in mobilen Spielen, *INFORMATIK 2004: Tagungsband der 34. GI-Jahrestagung, 20.9. – 24.9.2004, Ulm*, Vol. 50 of *LNI*, pp. 184–188.

[Hall and Anderson (2009)] Hall, S. P. and Anderson, E.: 2009, Operating Systems for Mobile Computing, *Journal of Computing Sciences in Colleges* **25**(2), 64–71.

[Hunicke (2005)] Hunicke, R.: 2005, The Case for Dynamic Difficulty Adjustment in Games, *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, pp. 429–433.

[Ingles (2006)] Ingles, B.: 2006, The Future of Java Game Development, *Proceedings of the 44th Annual Southeast Regional Conference*, ACM-SE '06, pp. 698–701.

[Kang (2009)] Kang, T.: 2009, Global Smartphone Vendor Market Share by Region: 2008, *Report*, Strategy Analytics.

[Krumm et al. (2008)] Krumm, J., Davies, N. and Narayanaswami, C.: 2008, User-Generated Content, *IEEE Pervasive Computing* **7**(4), 10–11.

[Magerkurth et al. (2005)] Magerkurth, C., Cheok, A. D., Mandryk, R. L. and Nilsen, T.: 2005, Pervasive Games: Bringing Computer Entertainment Back to the Real World, *Computers in Entertainment* **3**(3), 4:1–4:19.

[Paelke et al. (2004)] Paelke, V., Reimann, C. and Stichling, D.: 2004, Foot-based Mobile Interaction with Games, *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '04, pp. 321–324.

[Rollings and Adams (2003)] Rollings, A. and Adams, E.: 2003, *Andrew Rollings and Ernest Adams on Game Design*, New Riders Games.

[Schlieder et al. (2006)] Schlieder, C., Kiefer, P. and Matyas, S.: 2006, Geogames: Designing Location-Based Games from Classic Board Games, *IEEE Intelligent Systems* **21**(5), 40–46.

[Schmatz et al. (2009)] Schmatz, M., Henke, K., Türck, C., Mohr, C. and Sackmann, T.: 2009, SYGo – a location-based game adapted from the board game Scotland Yard, *INFORMATIK 2009: Tagungsband der 39. GI-Jahrestagung, 28.9. – 2.10.2009, Lübeck*, Vol. 154 of *LNI*, pp. 1891–1901.

[Taylor et al. (2006)] Taylor, M. J., Gresty, D. and Baskett, M.: 2006, Computer Game-flow Design, *Computers in Entertainment* **4**(1), 5:1–5:10.

[Tutzschke and Zukunft (2009)] Tutzschke, J.-P. and Zukunft, O.: 2009, FRAP: A Framework for Pervasive Games, *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '09, pp. 133–142.

[Wetzel et al. (2008)] Wetzel, R., McCall, R., Braun, A.-K. and Broll, W.: 2008, Guidelines for Designing Augmented Reality Games, *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, Future Play '08, pp. 173–180.

[Wolff and Grüter (2008)] Wolff, S. and Grüter, B.: 2008, Context, emergent game play and the mobile gamer as producer, *INFORMATIK 2008: Tagungsband der 38. GI-Jahrestagung, 8.9. – 13.9.2008, München*, Vol. 133 of *LNI*, pp. 495–500.