Dead on Arrival: Adapting Games to Finish at a Given Time or Location

Arne von Öhsen[1] and Jörn Loviscach[2]

[1] Hochschule Bremen (University of Applied Sciences), 28199 Bremen, Germany
v.oehsen@googlemail.com
http://www.oehsen.com
[2] Fachhochschule Bielefeld (Univ. of Applied Sciences), 33602 Bielefeld, Germany
jl@j3L7h.de
http://j3L7h.de

Abstract. Casual and other games often serve as time-killing applications, be it on the commuter train or in the back seat of a shared car. When one arrives at the destination, the game has to be interrupted or aborted, which is annoying or even frustrating. Hence, we propose to continuously adapt the game's level of difficulty to the estimated remaining time to arrival. This can be preset as a number of minutes or can continuously be estimated from the player's position in relation to a predefined destination. Our dungeon-style prototype is based on an automated engine for content placement and can also make use of GPS data. We report on preliminary results from user tests.

Keywords: game difficulty adaptation, automated content creation, GPS, location-based service, casual games.

1 Introduction

Typically, the notion of location-based gaming [1] refers to using actual buildings etc. as part of a game, for instance as a playing-ground for a tag game or as a backdrop for a fictional story. This paper proposes a novel type of incorporating time and/or location into a game: The game—an otherwise standard video game—is finished in a satisfying way after a preset time has elapsed or when the player arrives at a predefined destination. This is achieved by a continuous adaptation of the game's difficulty.

The adaptation requires creating and placing game content appropriate to the strength the user has shown so far. For the prototype, we created a dungeon-style first-person shooter game in which both the number of rooms and their content (including monsters to be killed) are variable. The content is arranged in the rooms through a rule-based process. Depending on the player's progress, the system removes rooms or adds rooms that have yet to be visited and creates appropriate content for them, see Figure 1. The last room contains the master monster to be conquered so that the game possesses a satisfying end.

The software comprises four major components: A regular 3D game engine with standard game logic serves as the front end. The rules to create and populate

A. Nijholt, D. Reidsma, and H. Hondorp (Eds.): INTETAIN 2009, LNICST 9, pp. 207–212, 2009.

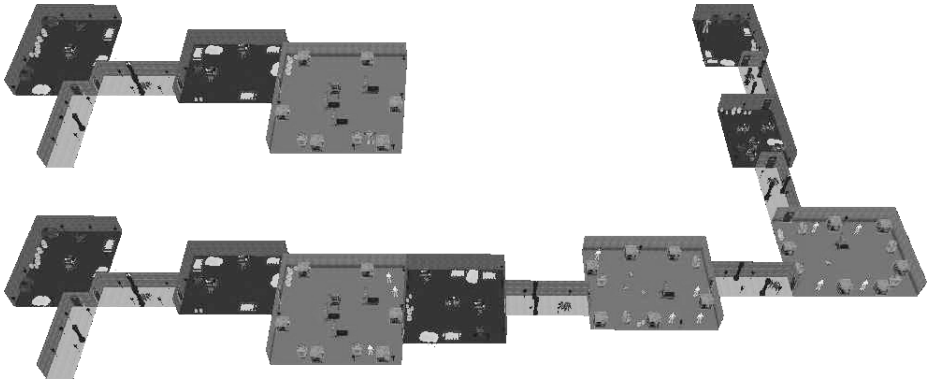

Fig. 1. To finish in the same time, a slower player will dynamically be presented with a simpler dungeon (top left) than will a quicker player (bottom and right)

rooms are stored in a database. A timing service provides an estimate on the remaining game time based on a direct setting by the user or on localization data obtained by a GPS device. These three components are tied together by the adaptation component that builds the gaming world and adapts it to the estimated remaining duration.

2 Related Work

Magerko [2] discusses domains in which game adaptation can be effective: storytelling (e. g., leading the player to decisions corresponding to the game author's intentions), player motivation (catering for different personalities and preferences), non-player character reasoning, and pedagogy. Most of these topics are connected to personalization [3], in contrast to the work presented here. For instance, Hunicke [4] describes a "dynamic difficulty adaptation" that intervenes to help the player's character stay at a given health level by placing helpful items or by manipulating the player's strength. This can already be perceived as a means to control the game's duration, namely to prevent from ending early.

Gustafsson et al. [5] describe another GPS-based adaptation of a game: Here, the game's story is adapted to the surrounding geographic items. Addressing a different type of entertainment by computers, Adcock et al. [6] introduce an audio player that tries to adapt the music selection to the available time.

3 Room Generation

The adaptation component employs an enhanced version of a system we introduced for automated placement of furniture [7]: A rule-based approach generates rooms, places immovable items in them and selects start positions and orientations for animated monsters. It operates on a collection of 3D content files such as models, textures, and animations that are described by a database. The entries of the database comprise low-level descriptions such as bounding boxes as

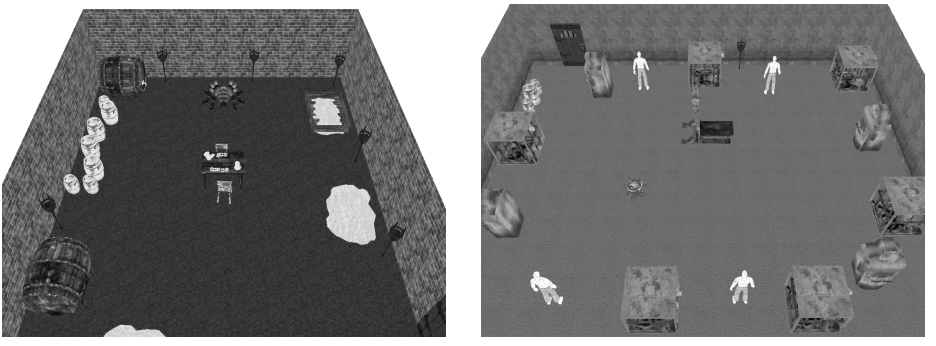

Fig. 2. Every room such as a lair (left) or a prison (right) is built and populated according to a set of rules associated with the room type and the items and monsters to be placed (bird's eye view, not available to the player)

well as high-level descriptions such as hierarchies and other relations between objects. The database entries for monsters are equipped with behavior scripts.

Rooms are not stored in the database in their finished form, but as collection of "room types," each of which comprises rules concerning the allowable size and frequency. Each room type possesses specific probabilities for the different kinds of items and monsters it may contain. To create a room, the system chooses among the room types obeying the given probabilities but not creating the same type in sequence. The room's size is determined at random. The room is attached to the rim of the existing dungeon with an appropriately placed door.

To populate a room, see Figure 2, a regular grid of about ten "room points" per square meter is created. These serve as hypothetical positions of items and initial positions of monsters. Items are picked according to their probability for the given room type. The placement process obeys the hierarchy prescribed in the database, which allows, for instance, to first place a table and then put the chairs around. Each item type is accompanied by rules in the database that specify how well such an item fits onto a given spare "room point." These rules can incorporate the vicinity of walls or other items of the same or other types. The optimum point is selected, proceeding from item to item in a greedy fashion.

The placement of the monsters proceeds in a similar fashion, with one difference: The ultimate room—i. e. the room entered when the time is about to expire—will always contain the master monster. Monsters in rooms that appear later in the game receive more life energy, meaning they are harder to kill. This increasing difficulty is intended to keep game more interesting. In a similar fashion, later rooms contain more loot, which can be picked up from the items and from dead monsters. It is distributed onto monsters and items alike.

4 Adaptation

On initialization, the system builds a dungeon of a size that would require a very good player to finish on time. If the player lags, the monsters in the ultimate room

are made weaker. If they become weaker than those in the penultimate room, the latter room is removed—if the player is not already in it—and the strength of the monsters in the ultimate room is incremented again. This allows a more flexible control than only changing the number of rooms. If the player is ahead of schedule, additional rooms have to be appended. This can happen in particular with location-based control. For instance, the player may wait for a bus during the beginning of the game, so that the estimated required time skyrockets.

To determine if the player is ahead of or behind schedule, the system estimates the time required to finish the game. This includes the time for the movement from door to door (distance times a general estimate of the velocity) and the time to kill the monsters (damage to be made divided by the estimated power of the player), which depends on player's and the monsters' life energies.

Depending on the user's choice, the remaining target time is determined through a simple timer or from location data. In the latter case, the distance between the location at the game's start and the current location is used to determine the average traveling velocity of the player in the real world. The estimate for the remaining time is computed by assuming that the player continues traveling at this velocity to the destination defined upfront.

5 Prototype and User Test

Our prototype is based on Mogre 1.4.8—a .NET wrapper of the game engine Ogre 1.4.8—using standard commercial and freeware dungeon props. The user's input is handled through the Object-Oriented Input System (OIS), wrapped by Mogre MOIS. The adaption is run once per second and takes about 15 ms on a standard PC; building a new room takes about 200 ms. For easy setup and editing, the databases are created as a collection of XML files. The localization control runs in a separate thread that is periodically queried by the main game thread. The position data are collected from a standard BlueTooth GPS receiver; currently, the end position is specified as latitude/longitude, which can be determined with help of a Google Maps script, for instance.

To get initial feedback on how well the adaptation works and on how invasive it feels we conducted a preliminary user test with 14 participants (12 m, 2 f; age 18 to 30). All subjects but one had several years of computer gaming experience; half of the participants reported to play ten or more hours per week. In the test, each subject played one session of the game after initial familiarization; afterward, he or she completed a questionnaire. For simplicity and reproducibility we chose the timer-based version of the prototype instead of the location-based version. The intended playing time was preset to three minutes. The participants were not informed upfront about the adaptation mechanism.

Almost all subjects indicated that the placement of the monsters and the items and the sequence of the rooms were either good or inconspicuous (13 of 14 for three questions). The majority of the participants found the end to fit into the game (11 of 14), but half of them reported that the game felt too simple.

Table 1 and Figure 3 show excerpts of statistics we collected during the game sessions. Even if the player's strength varies drastically, as demonstrated by the

Table 1. The statistics (examples from the test, sorted by playing time) indicate that the duration can be kept relatively constant. See the text for a discussion of the outliers

Playing Time	Rooms Visited	Monsters Killed	Score (Loot and Monsters)
2:04	3	1	0
2:59	13	21	658
3:00	9	14	2911
3:00	12	20	74
3:06	14	23	1428
3:09	6	8	2061
3:10	5	4	2424
3:19	9	13	327
3:21	9	14	2712
4:20	6	3	3215

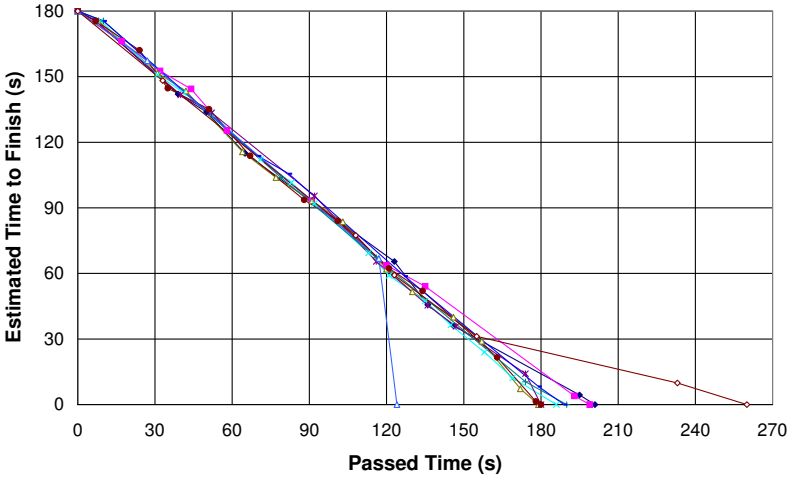

Fig. 3. Thanks to the repeated adaptation to the player's progress, the estimate on the remaining time remains closely tied to the eventual result (data of the same play sessions as given in Table 1)

scores, the game's duration only changes by some seconds. As the adaptation is not rigid, some fluctuation in the resulting duration cannot be avoided. We included the two most prominent outliers in the table: One user's character was killed early in the game; another user spent 77 seconds in the penultimate room, in which the game cannot be finished since this room was not scheduled to contain the master monster.

6 Conclusion and Outlook

We have demonstrated a method to let a video game finish at a certain time or when the player arrives at a given destination. The adaptation deals with a wide range of the users' abilities and remains mostly inconspicuous. We note,

however, some uncommon situations in which the room-based adaptation fails. Handling these would require substantial ad-hoc changes to the game's rules.

The current prototype can be used on a notebook computer with the adaptation being controlled through a GPS receiver. This does not cover all mobile settings, however. For instance, GPS data are not available on a subway train. Thus, a next step will be a version for a mobile phone that employs cell IDs for localization. Other options would be in-flight and (car) rear seat entertainment systems, where localization data or even estimated arrival times are readily available. These applications would require scaling up the gaming times—and thus the game's content and story—to accommodate for hours of playing. The basic mechanisms, however, could still be in place.

To create a version of the game with open portals instead of doors, one could limit the adjustments to parts of the world that have so far been invisible to the player. Finally, one could overcome the simplifying limitations of the dungeon and procedurally build—and continuously rebuild—a virtual city [8].

References

1. Benford, S., Magerkurth, C., Ljungstrand, P.: Bridging the Physical and Digital in Pervasive Gaming. Communications of the ACM 48(3), 54–57 (2005)
2. Magerko, B.: Adaptation in Digital Games. IEEE Computer, 87–89 (June 2008)
3. Wong, K.K.W.: Player Adaptive Entertainment Computing. In: 2nd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA), p. 13. ACM Press, New York (2007)
4. Hunicke, R.: The Case for Dynamic Difficulty Adjustment in Games. In: 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE), pp. 429–433. ACM Press, New York (2005)
5. Gustafsson, A., Bichard, J., Brunnberg, L., Juhlin, O., Combetto, M.: Believable Environments: Generating Interactive Storytelling in Vast Location-based Pervasive Games. In: ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE), Article No. 24. ACM Press, New York (2006)
6. Adcock, M., Chung, J., Schmandt, C.: AreWeThereYet? – A Temporally Aware Media Player. In: 9th Australian User Interface Conference (AUIC), p. 29–32. Australian Computer Society, Darlinghurst (2008)
7. Brauer, R., Öhsen, A., Loviscach, J.: Automated Interior Design from A to Z. In: SIGGRAPH 2008 Poster (2008)
8. Wonka, P., Hanson, E., Müller, P., Watson, B.: Procedural Modeling of Urban Environments. In: SIGGRAPH 2006 Course (2006)