Accelerometer and Spatial-Orientation Interfaces to Maze Games on Tablets and Mobile Devices

S.A. Innes, B.J. Kennedy, E. P. Clarkson, L.J. Edmonds and K.A. Hawick
Computer Science, Massey University, North Shore 102-904, Auckland, New Zealand
email: redmansteve2@hotmail.co.uk, benkennedy50@gmail.com
{elliot.clarkson,leuan.edmonds}@gmail.com, k.a.hawick@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

June 2013

ABSTRACT

Computer games on mobile platforms are increasingly popular and modern devices offer games developers unconventional user interface technologies based on device orientation. We describe a maze game implemented both for Android mobile phone and tablet devices that supports navigation though a generated landscape using tilt motions accessed via the devices accelerometer sensors. We report on implementation issues including device sensitivity, resolution and appropriate ways to utilise tilt motion in a practical game or simulation. We discuss future directions for this technology and possible uses in simulations as well as games.

KEY WORDS
computer games; mobile game; accelerometer; maze game; tilt navigation.

1 Introduction

Human Computer Interaction(HCI) [6] for computer games [27] and other highly interactive simulation or navigation programs is an important field with the potential to make use of new devices and interaction idioms for highly mobile devices such as tablet computers or mobile phones [8].

Interfaces [23] for modern computer games [17] can make use of the sensors such as tilt accelerometers that are now commonly available in both tablet and mobile phones. While touch screen interfaces are becoming widely used by various programs for tablets, uptake of the accelerometer sensor information has been slower. Touch sensitive devices have had a slow route to adoption [28] but the commodity pricing and now widespread availability of programmable devices such as Android or IOS enabled tablets or phones has led to an increasing market and demand for game Apps for such platforms.

A great deal of work has been reported on conventional HCI techniques [5, 24], but these new devices require new interaction metaphors and idioms that go beyond conventional graphical user interfaces [12]. There is considerable scope for new HCI frameworks [1] that can make use of development trends [10] in personal digital assistant(PDA) [25] technologies such as touch screens and other sensors.

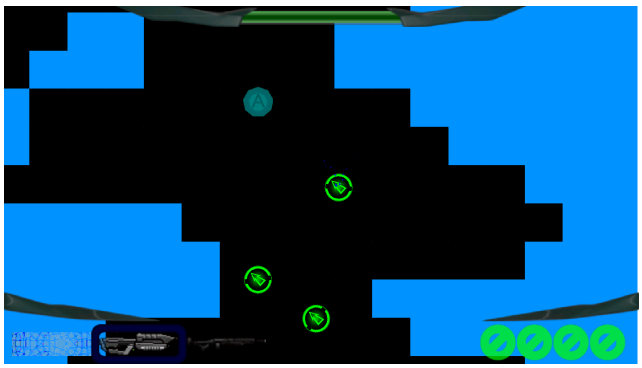

Figure 1: **Screen-shot of the prototype Androrbs game.**

Computer games are an important market for Apps that use these ideas [15], but there are also important applications areas such as education [16, 22] or training systems as well as more serious simulations [19, 20] and model demonstrators [18] that can also use them [9].

In this paper we describe how accelerometer sensor information can be used in a maze navigation game where the user tilts the whole device - tablet or phone - to make the player entity move around. Furthermore sensor fusion techniques can combine new touch gestures or multi-finger clicks [21] with the accelerometer orientation information to provide new game-player modes of interaction.

We also discuss a number of software architectural issues such as how to calibrate the tilting response to avoid the game or simulation appearing sluggish, and generation and manipulation of feasible maze maps that make use of the tilt sensor information.

Figure 1 shows a screen-shot from the Androrbs game showing three players within a generated map.

Tilt sensors [3, 13] are not new although they have only re-

cently become widely available in commodity proceed devices [2]. Work has been reported on use of tilt sensors for navigation of geophysical map data, health applications [14], floating raft navigation [26] and other serious applications [7] as well as games [11].

Our article is structured as follows: In Section 2 we describe how we used the accelerometer in various devices. We present some results concerning sensor control and sensitivity in Section 3 and discuss tablet orientation issues and comparable games on similar platforms in Section 4. In Section 5 we summarise our conclusions and offer some suggested areas for further investigation.

2 Architecture & Implementation

An accelerometer is an electro-mechanical device that measures acceleration forces. These forces can either be static, like the constant force of gravity, or they can be dynamic - caused by moving or vibrating the accelerometer. An accelerometer allows the device in which it is installed to have some idea about its own orientation and movement in space. Accelerometers measure the amount of static acceleration due to gravity to find out the angle the device is tilting at with respect to the earth. By sensing the amount of dynamic acceleration, it is possible to analyse the way the device is moving. The output of the accelerometer can help the device know if it is being dropped, tilted, or shaken.

Accelerometers have heavy use in many industries today. As well as being a staple addition to most tablets, the accelerometer sees use in laptops. Using the accelerometer a laptop is able to detect if it is falling and temporarily turns off the hard drive, protecting the head from crashing onto the platters. In a similar fashion, High-G accelerometers are the industry standard way of detecting car crashes and deploying airbags at just the right time. Because of the way most accelerometers are implemented, they suffer from high frequency noise.

In contrast to an accelerometer which measures acceleration, a gyroscope measures the rate of rotation around an axis and suffers from low frequency noise. Gyroscopes are useful because its output can be used to minimise the effects of noise from an accelerometer. Likewise, the output of an accelerometer can be used to minimise the long-term drifting effect that a gyroscope suffers from. This is known as sensor fusion and results in a more accurate output, which can greatly enhance the feeling of control by the user.

Accelerometer/Gyroscope sensor fusion is done automatically by Android since API level 9 [4] in the virtual sensors 'Linear Acceleration' and 'Gravity', but their presence depends on available hardware and device API level. Despite these sensors resulting in a clearer, more useful signal, they were not further examined due to a priority on compatibility with older devices.

2.1 Obtaining Accelerometer Data

Androrbs uses the AndEngine framework, which provides its own interface `IAccelerationListener`. However, this is entirely unnecessary and simply builds on Android's already existing sensor listeners.

Obtaining accelerometer data in Android 1.5 or above without a third party framework involves creating a `SensorManager`, obtaining a `Sensor` from it, and supplying that `Sensor` with a `SensorEventListener` instance to report its data to.

```
sensorManager = (SensorManager)
   getSystemService(Context.SENSOR_SERVICE);
Sensor asensor =
        sensorManager.getDefaultSensor(
                Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(
        this,
        asensor,
        SensorManager.SENSOR_DELAY_GAME);

@Override
public void onSensorChanged(SensorEvent ev)
{
    if (ev.sensor.getType() ==
            Sensor.TYPE_ACCELEROMETER)
    {
        Log.i("Accelerometer Example",
              "Accelerometer data: " +
               ev.values[0] + " " +
               ev.values[1] + " " +
               ev.values[2]);
    }
}
```

Figure 2: Obtaining acceleration data in Android Version 1.5 or later.

2.2 Movement Algorithms

```
final Vector2 velocity = Vector2Pool.obtain(
        pAccelerationData.getX() * 5,
        pAccelerationData.getY() * 5);
playerBody.setLinearVelocity(velocity);
Vector2Pool.recycle(velocity);
```

Figure 3: Initial movement algorithm.

Figure 3 shows the first attempt to control the player entity, by simply applying the forces obtained from the accelerometer directly as velocity. This results in a mathematically incorrect but extremely responsive control scheme. Because acceleration is completely bypassed and a force is never applied, the entity responds as quickly as the device can be tilted. This feels bizarre, but not entirely unpleasant. We

felt this was not appropriate for our game, but interesting enough to note.

```
Vector2 f = Vector2Pool.obtain(
        pAccelerationData.getX() * accel,
        pAccelerationData.getY() * accel);
orb.getBody().applyForce(f,
        orb.getBody().getWorldCenter());
Vector2Pool.recycle(f);
```

Figure 4: Secondary movement algorithm.

Figure 4 shows the second attempt at moving the player entity via a function of accelerometer output. This time, actual forces are applied to the player entity to move it. This is the most mathematically accurate algorithm and the entity responds as it would in reality if the equivalent forces were applied. However, mathematical accuracy does not necessarily guarantee a pleasant experience. The entity seemed unresponsive, and felt out of place for a fast-paced game such as Androrbs.

```
accel = 5.0f, counterAccel = 4.0f.
Vector2 f = Vector2Pool.obtain(
        pAccelerationData.getX() * accel,
        pAccelerationData.getY() * accel);
Vector2 v = Vector2Pool.obtain(
        orb.getBody().getLinearVelocity());
// If we are trying to accelerate in an
// opposite direction to movement, compensate
if (f.x * v.x < 0)
        f.x -= v.x * counterAccel;
if (f.y * v.y < 0)
        f.y -= v.y * counterAccel;


orb.getBody().applyForce(f,
        orb.getBody().getWorldCenter());
Vector2Pool.recycle(v);
Vector2Pool.recycle(f);
```

Figure 5: Movement algorithm 3

Figure 5 shows the third iteration of the movement algorithm. To try to compensate for the unsatisfying motion of the previous code iteration, if the player tries to accelerate in the opposite direction to their current motion, they receive a boost to their newly applied force by their current velocity magnified by a 'counter-acceleration' constant.

Figure 6 shows an updated version of the movement algorithm with much more accurate calculation of friction.

2.3 Interfacing with the game

We wanted the game to behave ("feel") like a ball bearing puzzle (Figure 7), in which you navigate ball bearings

```
final static float friction=0.3;
Vector2 f = Vector2Pool.obtain(
        pAccelerationData.getX() * accel,
        pAccelerationData.getY() * accel);
Vector2 v = Vector2Pool.obtain(
        orb.getBody().getLinearVelocity());
f += friction*-v; // Apply friction

orb.getBody().applyForce(f,
        orb.getBody().getWorldCenter());
Vector2Pool.recycle(v);
Vector2Pool.recycle(f);
```

Figure 6: Movement algorithm 4

Figure 7: **A ball bearing puzzle game, must be tilted to solve.**

through a maze to a desired location by tilting the board. The maze is randomly generated in our levels see 2.6 for details. The tilting gesture is used in the game to move the players "orb", allowing the user to navigate a maze and attack opposition players.

2.4 How it is implemented

Android being the Operating System of choice requires the use of the Java programming language for applications. The open source game engine AndEngine was used to build our game upon. This gave us easy access to the Box2D physics engine and interfaced with the Android motion sensors. AndEngine scales the game to different resolutions automatically so no extra work was required to make the game work correctly on a tablet or mobile phone.

2.5 Accelerometer limits

Accelerometers give us an interesting method of input for a tablet or phone device, but when using such methods of control, there are considerations one should take into account

with regards to the resultant simulated motion of player entities. Unlike using a directional pad or on-screen control method, the input cannot change instantaneously from one extreme to another.

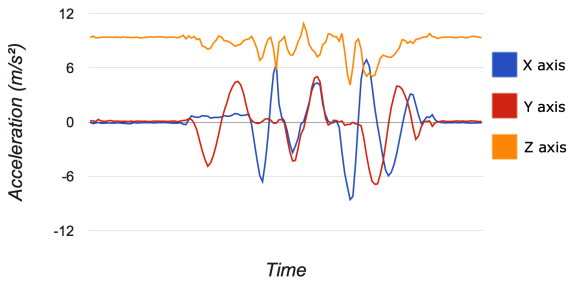

Figure 8: Smoothing of player intent signal by accelerometer over a period of two seconds.

Instead, input is always a gradual arc, as demonstrated in Figure 8. The smoothing of the original signal of player intent becomes more destructive as their input device is heavier, and gains more momentum with motion. This effect is magnified over long use periods, as the natural inclination of the player is to use a more relaxed grip.

Care should be taken to account for this on a game design level.

2.6 Maze generation

Because of the accelerometer limits discussed in the previous section, some environments could prove difficult to navigate. Unless the challenge is specifically to navigate difficult terrain, it may be wiser to provide players with natural, sweeping terrain, which mimics the input gathered from the accelerometer and levels the playing field for those on larger tablet devices.

There are a variety of signal processing techniques which could be applied to existing landscapes to improve their appropriateness for traversal with an accelerometer or similar device. Here, a cellular automata approach will be discussed for procedural generation of map data. One of the most famous rule tables for cellular automata is Conway's Game Of Life due to the highly dynamic patterns that can emerge. There are other rule sets in existence which are less dynamic and unpredictable which can be used to generate interesting structures. One such rule is the 4-5 rule, valid for any 2 dimensional grid of wall and floor tiles. This rule is interesting because after a few iterations on a grid filled with a random selection of floor and wall in some proportion as a function of cavern density, natural and organic flowing form emerges. This rule is described thusly, using a self-inclusive Moore neighbourhood:

- A tile becomes a wall if:
  - It was previously a wall and 4 or more of its 9 neighbours were walls
  - It was previously not a wall and 5 or more of its 9 neighbours were walls
- Otherwise, the tile becomes a floor.

This rule is capable of generating organic, cave-like structures of any size, as demonstrated in Figure 9.

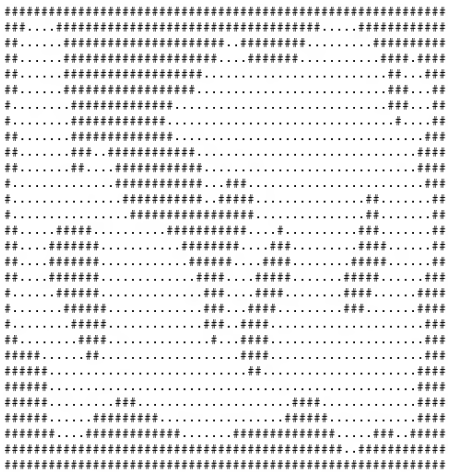

Figure 9: A sample generated map using the 4-5 rule. Note the flowing, natural curves.

2.7 Map Optimization

Sometimes the map generator would produce a map that was not particularly fun to play. These include maps almost entirely made of wall or floor, and maps with disconnected, unreachable segments. To deal with these situations, some checks are done before the map is selected for play. If any of these checks fail, the map is discarded, and another one is produced. This ensures entertaining maps every time.

First, disconnected areas must be dealt with. There is no guarantee that all areas of the map will be accessible by the player, and if care is not taken with start point selection, players may find themselves trapped in a small area. The approach taken here was to fill all but the largest connected area of floor with wall, effectively removing any smaller pockets that may have emerged. However, this reduces the number of floor tiles present in the map. It is recommended that after this elimination pass is complete, the number of floor tiles is evaluated, and the map discarded and regenerated if the proportion of wall to floor falls below a certain threshold.

While this approach does generate smooth curves, it does so in a rather uncontrolled manner. It is, however, possible to 'seed' the generated structure to adhere to a rough design. A technique considered was to generate a small maze with another algorithm such as Prim's algorithm or a simple

Depth-first search. The generated tiles were then scaled up, so that for each tile of output, an area of 7x7 random tiles are created, with probabilities weighted towards the original tile type. The 4-5 rule is then run on this field. Because of the non-random input to 4-5, the resulting output is fitted to the weighted maze probabilities, biasing the system to following the previously generated structure.

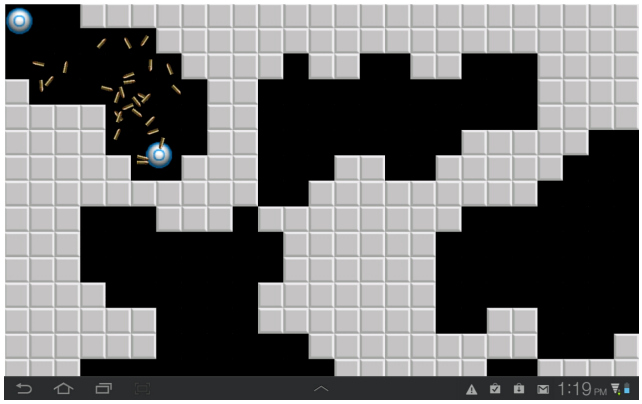

Figure 10: The disconnected map problem.

Note that this approach becomes significantly more complicated when the player entity is larger than one tile, as it is difficult to guarantee that all areas of the generated landscape are accessible. When the player entity is large, choke points could emerge - Segments of the landscape which are too narrow for the player to pass through, yet mean that the map is still actually connected. These are difficult and expensive to detect, so generating fine resolution landscapes this way comes at a downside.

3 Experimental Results

There are various aspects of the tilt sensor behaviour that arise from the user example as shown in Figure 11.

The first iteration of the movement algorithm (applying the forces of the accelerometer directly as the velocity of the player) proved very effective. The movement was alarmingly responsive. Unfortunately the movement proved too responsive; the player entity moving much too fast for a user to accurately control in the close quarters of the maze, most of the time leaving the user crashing into walls. In order to counteract this the speed was modified in order to make the player orb easier to control. Because the player orb only used velocity for movement, no matter how the speed was adjusted the motion did not seem natural.

The second iteration of the movement algorithm used the acceleration given by the accelerometer to equate force to apply to the orb. This lead the user to have much better control over the player orb, however turning became an issue without the snappy reactiveness of the other algorithm. As speed was a factor in Androrbs, the algorithm needed to be improved upon to optimise game-play.

The third iteration of the movement algorithm built upon the second by utilising a ‘counter-acceleration’ constant. This had the effect of amplifying movements made in the opposite direction of which the user was originally traveling. This brought back the reactiveness of the first algorithm while allowing the speed of the player orb to not get out of hand.

The fourth iteration of the movement algorithm is a finely polished version of the third. It gave the user enough speed to allow them to run around the maze easily without having to hug a wall. It also gave the user the right amount of control in order to hide in the maze and traverse some of the more narrow passages that might develop.

Because of the high frequency noise the accelerometer is exposed to, there is a decent amount of extra input the user does not deliberately make. This became evident in development as the player orb would casually roll to a corner when placed on a flat table. Occasionally, it would also shuffle slightly in one position when balanced in the middle of the screen. In order to stop this from happening the algorithms were changed in order to filter out small inputs and eradicate noise. This proved to be successful but made the player orb harder to control in tight spaces, eliminating the small degrees of motion needed to accurately roll through. This decision was later revoked in order to keep the player orb as easy to control as possible. It was also evident that as the main control scheme was the tilting motion of the accelerometer, and the game was designed to be fast paced, there would not be many times a user would want to remain perfectly still, finding it much more beneficial to roll away as fast as possible than staying still and hoping not to be found.

As mentioned previously, the accelerometer suffers from high frequency noise. Steps were taken in order to minimize this and improve the overall stability of the accelerometer, but overall this hindered game-play by restricting the finer movements. Inputs differ from devices depending on how well the accelerometer is made. If the Android device contains a gyroscope this can be remedied with the help of sensor fusion. If the device contains both a gyroscope and an accelerometer the input is smoothed out, removing the majority of noise, allowing for a much more stable signal and a much more pleasing experience. On development of the application, phones with gyroscopes were not as common and most could not utilise sensor fusion, making the

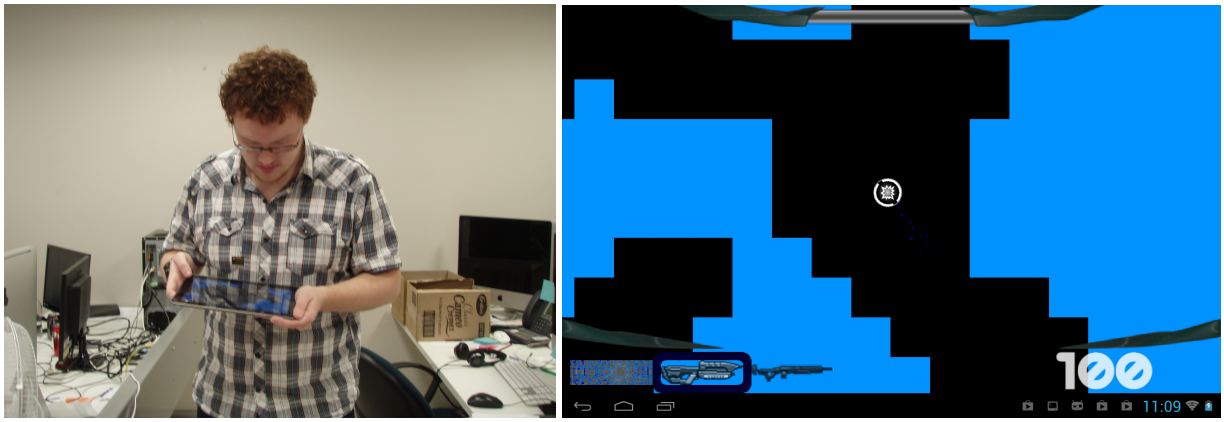

Figure 11: Tilting the top left corner of the tablet resulting in a movement towards the top left corner of the in-game screen.

tablet-controlled versions of the App a lot more enjoyable with the controls feeling a lot more stable.

4 Discussion

The control scheme of the game needed the player to keep the tablet mostly flat and held above the ground. This was not very intuitive to new players and needed to be explained before the game could be played properly. If the game was to be developed further, a useful option would be to calibrate the game with a default orientation. This would allow a player to hold the device more comfortably in front of them, rather than having to keep it flat and below the head, which may prove to be uncomfortable for long periods of time. This may also prove to be counter-productive as it may be confusing for a user to comprehend the adjusted orientation. It is easy to think about moving a surface with your hands and having the ball roll down it, but if that surface was in front of the player horizontally it would effectively make the player have to grasp the concept of rolling up a wall.

4.1 Other games and platforms

A game similar in concept to ours is "MadBalls in Babo Invasion" in which the player controls spherical characters that resemble the "MadBalls" toys. The game is almost identical to our own in that the players move through a maze using an orb like character in order to find and defeat other players who are trying to accomplish the same of you. This game is a lot more graphically impressive than ours as it has been commercially made with a much wider development team. The game is developed for the PC which had a lot more graphical power behind it when the game was made. The game uses the 'WASD' control scheme on a keyboard. Comparatively, Androrbs seems to be more natural to control than 'MadBalls' as it feels more physically accurate using an accelerometer-based control scheme. That being said, the 'WASD' control scheme does not suffer from the high frequency noise problem that plagues the accelerometer, leading to a much more accurate sense of control.

Because the App was written in Java it would be very simple to move it from one platform to another. The main challenge would be replacing the components we used from the AndEngine to suit another platform. When developing the App, one of the main goals was to make something that both older and newer devices could play. Because so many mobile platforms come equipped with an accelerometer as standard, the App can be played with a plethora of devices.

5 Conclusions

We made an App that uses the accelerometer on an Android device, a standard in the majority of mobile platforms, as its main control scheme in order to cater for the broadest market possible. We took the limits of the input device into account when designing traversable landscapes using a cellular automata, and discussed methods of map optimization such as initial seeding with non-random values and avoidance of disconnected areas.

We found that using the accelerometer as a control scheme is not only simple and easy to do, but also relatively accurate. This accuracy is further improved with that of a gyroscope. The developer can use sensor fusion in order to smooth out the noise of the accelerometer. Accelerometer movement can be considered superior to traditional methods of input in terms of intuitiveness and ease of use, but may be tiring for prolonged play.

We believe the accelerometer could be used to make multiplatform games with little to no hassle, provided its limits are respected and elements of the game which closely interact with accelerometer data are designed appropriately.

Whenever a correctly configured sensor-fused virtual input is available which makes use of the accelerometer as an input, it will always provide a less noisy signal than the ac-

celerometer alone. There is scope for further research on exactly how this newfound accuracy could be leveraged. In addition, the z-axis is largely ignored in our investigation, and could be used to provide additional input.

References

[1] Ali, S.I., Jain, S., Lal, B., Sharma, N.: A framework for modeling and designing of intelligent and adaptive interfaces for human computer interaction. Int. J. Applied Information Systems 1(2), 20–25 (January 2012)

[2] Browne, K., Anand, C.: An empirical evaluation of user interfaces for a mobile video game. Entertainment Computing 3, 1–10 (2012)

[3] Chatting, D.J.: Action and reaction for physical map interfaces. In: Proc. Second Int. Conf. on Tangible and Embedded Interaction (TEI'08). pp. 187–190. Bonn, Germany (18-20 Feb 2008)

[4] Developers, A.: Software development kit (February 2012), http://developer.android.com/sdk, androind SDK

[5] Diaper, D., Stanton, N. (eds.): The Handbook of Task Analysis for Human-Computer Interaction. IEA (2004)

[6] Dix, A., Finlay, J., Abowd, G., Beale, R.: Human-Computer Interaction. Prentice Hall (1993)

[7] Elmaghraby, A., Mendez, A., Zapirain, B.G.: Serious games and health informatics: A unified framework. In: Proc. 17th IEEE Int. Conf. on Computer Games (CGAMES 2012). pp. 35–38 (2012)

[8] Furio, D., Gonzalez-Gancedo, S., Juan, M.C., Segui, I., Costa, M.: The effects of the size and weight of a mobile device on an educational game. Computers and Education 64, 24–41 (2013)

[9] Gerdelan, A.P., Hawick, K.A., Leist, A., Playne, D.P.: Simulation frameworks for virtual environments. In: Proc. International Conference on Internet Computing (ICOMP'11). pp. 272–278. No. ICM4087, CSREA, Las Vegas, USA (18-21 July 2011)

[10] Gouin, D., Lavigue, V.: Trends in human-computer interaction to support future intelligence analysis capabilities. In: Proc. 16th Int. Command and Control Research and Technology Symposium. No. Paper 130, Quebec City, Canada (21-23 June 2010)

[11] Hwang, I., Lee, Y., Park, T., Song, J.: Toward a mobile platform for pervasive games. In: Proc. ACM MobiGames'12. pp. 19–24. Helsinki, Finland (2012)

[12] Kortum, P.: HCI Beyond the GUI - Design for Hapric, Speech, Olfactory and other Nontraditional Interfaces. Morgan Kaufmann (2008)

[13] MacKenzie, I.S., Teather, R.J.: Fittstilt: The application of fitts? law to tilt-based interaction. In: Proc. 7th Nordic Conference on Hman-Computer Interaction (NordCHI'12). pp. 568–577. Copenhagen, Denmark (14-17 October 2012)

[14] McCallum, S.: Gamification and serious games for personalized health. In: Proc. 9th Int. Conf. on Wearable Micro and Nano Technologies for Personalized Health (pHealth'12). Porto, Portugal (26-28 June 2012)

[15] Muehl, W., Novak, J.: Game Development Essentials - Game Simulation Development. Delmar (2008), iSBn 978-1-4180-6439-6

[16] Narayan, M.A., Chen, J., Perez-Quinones, M.A.: Usability of tablet pc as a remote control device for biomedical data visualization applications. Tech. Rep. TR04-26, Virginia Tech (2004), http://eprints.cs.vt.edu/archive/00000703/01/TR04-26-Jian-Michael.pdf

[17] Novak, J.: Game Development Essentials - An Introduction. Delmar, 3rd edn. (2012)

[18] Pearce, B.T., Hawick, K.A.: Interactive simulation and visualisation of falling sand pictures on tablet computers. In: Proc. 10th International Conference on Modeling, Simulation and Visualization Methods)MSV'13). p. MSV2341. No. CSTN-196, WorldComp, Las Vegas, USA (22-25 July 2013)

[19] Preez, V.D., Pearce, B., Hawick, K.A., McMullen, T.H.: Human-computer interaction on touch screen tablets for highly interactive computational simulations. In: Proc. International Conference on Human-Computer Interaction. pp. 258–265. IASTED, Baltimore, USA. (14-16 May 2012)

[20] Preez, V.D., Pearce, B., Hawick, K.A., McMullen, T.H.: Software engineering a family of complex systems simulation model apps on android tablets. In: Proc. Int. Conf. on Software Engineering Research and Practice (SERP'12). pp. 215–221. SERP12-authors.pdf, CSREA, Las Vegas, USA (16-19 July 2012)

[21] Rautaray, S.S., Agrawal, A.: Real time multiple hand gesture recognition system for human computer interaction. Int. J. Intelligent Systems and Applications 5, 56–64 (2012)

[22] Reed, R.H., Berque, D.A. (eds.): The Impact of Tablet PCs and Pen-based Technology on Education. Purdue Univ. press (2010)

[23] Saunders, K.D., Novak, J.: Game Development Essentials - Game Interface Design. Delmar, 2nd edn. (2013), iSBN 978-1-111-64288-4

[24] Scogings, C.J.: The Integration of Task and Dialogue Modelling in the Early Stages of User Interface Design. Ph.D. thesis, Massey University (2003)

[25] Tesoriero, R., Montero, F., Lozano, M.D., Gallud, J.A.: Hci design patterns for pda running space structured applications. In: Proc. 12th Int. Conf. on Human-Computer Interaction: Interaction Design and Usability (2007)

[26] Tomlinson, B., Yau, M.L., O'Connell, J., Williams, K., Yamaoka, S.: The virtual raft project: A mobile interface for interacting with communities of autonomous characters. In: Proc. ACM CHI '05. pp. 1150–1151. Portland, Oregon, USA (2-7 April 2005)

[27] Turner, J., Browning, D.: Workshop on hci and game interfaces: A long romance. In: Proc. OZCHI 2010 : Design, Interaction, Participation. Queensland University of Technology, Brisbane, Queensland, Australia (22-26 November 2010 2010)

[28] Yarow, J.: Tablet computing: A history of failure. Business Insider Online, 1–3 (January 2010)