Arkan∞id

Breaking Out of a Finite Space

Nels E. Beckman

School of Computer Science
Carnegie Mellon University
nbeckman@cs.cmu.edu

Abstract

Brick breaking games such as Breakout and Arkanoid have existed for years. However, they have all continued to propagate one simple design flaw, possibly in the name of "fun." We call this mistake, "finity." Each level contains a finite number of bricks to break, and there are only a finite number of levels. If in fact our paddle is a spaceship, and the entire game is taking place in outer-space, as Arkanoid's cryptic story would have us believe, then why is this universe finite? Physicists have come to believe that the universe is infinite, or at least really really big. Therefore, in this paper we present Arkan∞id, the infinite brick breaking game, a game in which the breaking of bricks will always reveal more bricks.

Categories and Subject Descriptors K.3 [*Computing Milieux*]: Computers and Education

General Terms Theory, Performance, Legal Aspects

Keywords Breakout, Arkanoid, Space, Paddle, Infinity, Boring

1. Introduction

On May 13, 1976 Atari, Inc. shocked the world when they released *Breakout*. Originally conceived as a single-player version of their popular *Pong* title, in *Breakout* the player controls a single paddle at the bottom of the screen, which they can move along the horizontal axis. The goal of the game is to use the paddle to deflect a bouncing ball upwards so that it will hit and thus destroy the many bricks tiled at the top of the screen. After all of the bricks are destroyed, the player proceeds to the next level.

In 1986 the Taito Corporation released a follow up game which revealed that what we had assumed up until that point was a mere paddle was in fact a spaceship. This spaceship, known as the "Vaus," had escaped from the doomed mother-ship, the "Arkanoid," the title of the game. The full story was presented in the opening dialog:

> THE ERA AND TIME OF THIS STORY IS UNKNOWN. AFTER THE MOTHERSHIP "ARKANOID" WAS DESTROYED, A SPACECRAFT "VAUS" SCRAMBLED AWAY FROM IT. BUT ONLY TO BE TRAPPED IN SPACE WARPED BY SOMEONE.....

For a long time this proved satisfactory. Numerous sequels, authorized and otherwise, were developed over the years, each of which explored the Arkanoid mythology in their own way. However, none these works diverged from the basic game-play model of the original Breakout game. At some point, skeptical scientists began asking difficult questions. For one, why was it necessary to tell the story in all caps? Also, is it not the case that that the standard ellipsis uses three periods rather than eight? But the most vexing issue, and the question addressed by this work, is the following: If, as physicists tell us, space is truly infinite, why does the game take place in a series of levels each consisting of a finite number of bricks? Should there be an infinite number in each level? Moreover, is not the very notion of levels inconsistent with an infinite universe?

We have concluded that the original Breakout and the later Arkanoid consisted of a finite number of bricks and levels only because of technical limitations of the era. Follow-up games from later time periods mistakenly assumed that the finite nature of the earlier works was to be emulated. Arkan∞id rectifies this long-standing error.

Figure 1. Initially, Arkan∞id looks like any other Breakout clone.

Figure 2. When the lowest screen of bricks has been broken, the ball moves up to a higher level of bricks and the camera follows it, ad inifitum.

Arkan∞id is an *infinite* game of outer-space brick breaking. While the basic game-play is the same as games like Arkanoid, the number of bricks that the player can destroy is effectively limitless. At all times more bricks await just off-screen. Arkan∞id is a game written in Java for the Blackberry mobile phone platform, and in this paper we describe its design and implementation.

2. Arkan∞id

In this section we briefly describe the game-play of Arkan∞id. Initially, Arkan∞id plays just like any other Breakout clone, as you can see in Figure 1. A ball is put into play and destroys each brick with which it collides. However, things get interesting once the first screen-full of bricks is eliminated. The camera scrolls to reveal more bricks, as shown in Figure 2. When those bricks are destroyed, the camera scrolls to reveal yet more bricks. This same behavior continues to occur until either a.)the player is bored or b.) the player's cell phone runs out of batteries.

While this may sounds incredibly boring, you are wrong. Only an infinite number of bricks could accurately represent the vast cosmos. We have done this for science not for you entertainment. What have you done lately for science?

Arkanoid is currently available for free download[1] on your Blackberry mobile device.

3. Implementation

In this section we describe the implementation of Arkan∞id, starting with a discussion of the design space.

3.1 Design Considerations

During the earliest stages of the design of Arkan∞id, there were certain goals that we wanted to accomplish. Here we briefly describe them.

The first and most important constraint was that, if the plane of bricks was to scroll on indefinitely, we did not want the amount of memory used to increase along with it. The Blackberry platform has a relatively small heap, and we could hardly say that Arkan∞id ran on

[1] `http://a8.nelsbeckman.com/`

forever if someone were able to leave the game running for a few days and witness a heap overflow.

Which brings us to the second point. Due to the infinite nature of our game, we would like gamers to be able to "play" Arkan∞id even when they cannot focus their attention on their phone. Therefore, our game should somehow continue even when the user is, say, talking on their phone, sleeping, or lifting weights. All this should potentially occur with some loss of score, so that players who decide never to rest, socialize or exercise can be known as "hard-core."

Finally, due to the decreased processing capabilities of the Blackberry platform, we must ensure that Arkan∞id is reasonably performant. This implies that we must somehow scale down the number of bricks for which collision detection will be performed, and the number of bricks that are given to the underlying platform to draw. If we naively perform collision detection on ever single brick, or even just every single brick that we have seen so far, we are doomed to poor performance.

All of these factors lead to the eventual design.

3.2 The Implementation of Arkan∞id

Arkan∞id is implemented in the Java programming language. The entire implementation of Arkan∞id centers around the BrickBoard class. The BrickBoard represents the infinite number of bricks that may exist in the game. Naturally, these bricks are created in an on-demand fashion, and we attempt to remove any overhead from bricks that have been destroyed.

A BrickBoard is essentially a linked-list of arrays. Its internal structure, which we will now describe, is illustrated in Figure 3. Each element in the linked-list holds a byte array of size thirteen. One such array can be used to represent one screen-full of bricks, if we use just one bit (on/off) to represent a brick. Each screen-full also has a pointer to the screen-full before it and after it, as well as a 64-bit integer holding the logical position on the y axis of the bottom of that particular screen-full. The next pointer of the last screen-full will always be set to `null`. Gradually, the linked list of brick arrays will be extended, one screen-full at a time, by dynamically allocating a new array, to go at the end of the list. However, this action will only be performed the first time that the ball passes into the logical coordinate space beyond the last allocated screen-full. In this way, we will only allocate memory to represent the bricks when necessary.

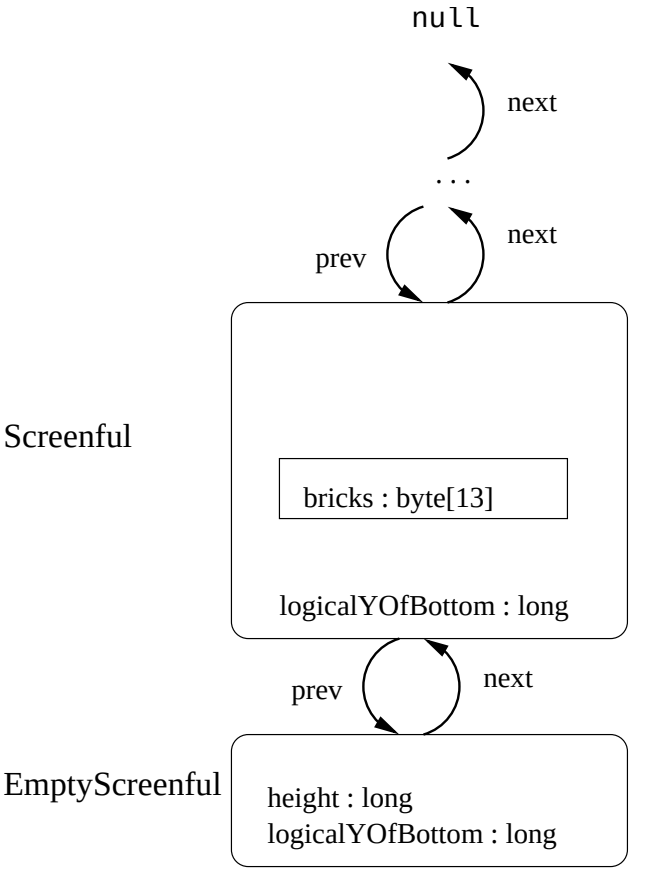

Figure 3. The design of the BrickBoard class, which holds a linked list of byte arrays, each of which represents a screen-full of bricks. At the bottom, one object represents all of the "empty space" at the bottom of the stack of bricks.

Still, if we were using an object to hold every screen-full of bricks, even a very small object, eventually we would exhaust our heap space, therefore spoiling the illusion of infinity. For this reason, the BrickBoard occasionally performs a "garbage collection" operation, during which the number of objects required to represent all screen-fulls empty of bricks will be reduced to a constant number. Whenever garbage collection is performed, starting from the beginning of the linked list, we collapse all screen-full objects whose arrays contain all zeros into one object of type EmptyScreenful. While it is possible for one brick to prevent a larger number of screen-fulls from being garbage collected, in practice we found that all lower bricks will eventually be eliminated, thus freeing up the available object.

We also have to find a way to reduce the number of bricks upon which collision detection will be performed. In Arkan∞id we track the ball with a camera, which ensures that the ball could only ever possibly collide with bricks that are on-screen. Therefore, our implementation will simply find the two screen-fulls of bricks that could possibly be on screen and only draw and perform collision detection on those two screen-fulls of bricks.

In practice, this design of the BrickBoard class allows for excellent performance and the ability for infinite bricks to exist with only a constant run-time overhead.

Finally, in order to allow gamers to play without actually devoting all of their attention to the cell phone, there is no limit on the number of balls that can be lost. Each time a ball goes past the paddle, a new ball is put into play, although we keep track of the number of balls the player has lost. Hardcore gamers will invariably brag about the low number of balls that they have lost while playing Arkan∞id and berate other "n00bs" for their inferior skillZ.

4. Discussion

In the end, it is worth discussing whether or not Arkan∞id really is an *infinite* game of brick breaking. Clearly it can last a long time, but forever? Given that we only use a constant amount of overhead to represent the bricks (one object for all of the empty screen-fulls and a some small number for the bricks that have not yet been destroyed) the only true limit that we must be wary of the position on the y axis of the ball. Since we are using a 64 bit integer to represent this position, we must consider the fact that eventually this integer will overflow, resulting in untold havoc. Let us consider how long we can play before this happens.

In Arkan∞id the ball will always move at a fixed vertical velocity, 5 pixels every 50 milliseconds. Our initial implementation used a 32 bit integer to represent the ball's vertical position. However, according to the following math;

$$\frac{2^{32}\text{ pixels}}{\left(\frac{5\text{ pixels}}{.05\text{ s}}\right)} \times \frac{1}{60\ \frac{\text{s}}{\text{m}} \times 60\ \frac{\text{m}}{\text{h}} \times 24\ \frac{\text{h}}{\text{d}}} = 497 \text{ days}$$

a ball moving upwards would only need 497 days of play time to overflow a 32 bit integer. Clearly this is a limitation in the implementation that could be observed. It was thus that we decided to use a 32 bit integer. This way, and according to the following math;

$$\frac{2^{64}\text{ pixels}}{\left(\frac{5\text{ pixels}}{.05\text{ s}}\right)} \times \frac{1}{60\ \frac{\text{s}}{\text{m}} \times 60\ \frac{\text{m}}{\text{h}} \times 24\ \frac{\text{h}}{\text{d}} \times 365\ \frac{\text{d}}{\text{y}}} =$$

$$58 \text{ Billion years}$$

58 billion years is certainly much more respectable. For all we know, 58 billion years might be an infinite amount of time. I have not yet been able to prove otherwise. As you may notice, all of the numbers represent just the amount of time it takes for the ball to go from the absolute bottom of the logical y coordinates to the absolute top. In reality, in order to get all the way to the top of the logical coordinates, the ball will have to go up and down an extremely large number of times so as to knock out earlier bricks. Therefore the actual amount of play time will be much greater than 58 billion years.

However, for math nerds who claim that we can do better, we have saved a tasty slice for future work. Eventually, we plan to use Java's BigInteger, an object that holds integers of arbitrary precision, to represent the logical address of the ball in the y coordinate plane. Then the maximum height the ball could reach would only be limited by the amount of heap space on your device. Since my Blackberry Pearl has 40MB of heap space, this means we could address approximately $256^{40,602,000}$ before we would have to stop playing. Unfortunately, such a large number causes my calculator to overflow. Sad face. So I am going to assume that it would be a really long time.

5. Conclusion

In this paper we presented the design and implementation of Arkan∞id, the realization of a line of work that began with 1976's Breakout. We argue that since the universe is infinite, or at least really really big, the idea of an outer-space brick breaking game in which there are only a finite number of bricks is patently ridiculous. While early brick breaking games were limited by technological constraints, we believe that later brick breaking games mistakenly copied this deficiency for reasons of "fun." With Arkan∞id have remedied the situation, creating an outer-space brick breaking game that need never end. We expect this sort of infinite playability to become the norm in future games. Earth, you're welcome.