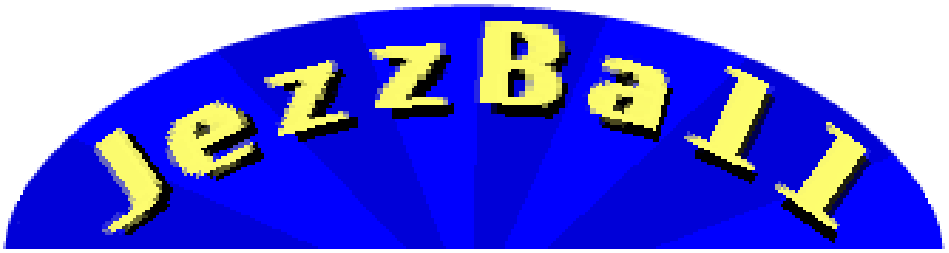


© 2025 Piotr Kowalewski

JezzBall is an arcade-style game for the HP Prime, written primarily in Python with a few snippets of PPL code.

The objective is to fill at least 75% of the rectangular playfield by launching splitters that divide the space. Balls bounce around the area, and if a splitter comes into contact with a ball while extending, the splitter is destroyed and you lose a life. When a splitter successfully divides the area into two parts, the region without any balls gets filled, bringing you closer to your goal.

Splitters extend horizontally or vertically from the selected point until they reach a wall or collide with a ball. Only one splitter can be active at a time – you must wait for it to finish extending before launching another.

Each level adds more balls, and you always have one more life than the number of balls. Splitters also become slightly faster with each level.

The shot counter tracks how many splitters you've launched during the level, while the fill indicator shows the percentage of the area filled. A yellow vertical line on the indicator marks the threshold required to complete the level.

Your score is based on the area filled. The larger the section filled in a single move, the more valuable each unit of that area becomes—rewarding efficient and well-timed splits. Additionally, the value of each filled unit increases with every level, so scoring opportunities grow as the game progresses.

Tip:

On the G2, before launching JezzBall, it is recommended to increase the screen refresh rate to ensure smoother animation (be sure to avoid the lowest setting). To adjust the refresh rate, press and hold the [On] key, then press and hold the [Shift] key, and finally press the [+] key several times (a bar indicator will appear showing the current value). Release all keys when done.

The HP Prime uses an LCD panel, which can sometimes cause motion blur during fast animations. If you notice the balls looking stretched or oval-shaped, try looking at the screen straight on – this can reduce the effect and make the image clearer.

Controls:

- **Arrow keys** – Move the splitter
- **Symb** – Change splitter orientation
- **Apps** or **Enter** – Launch the splitter
- **Esc** – Quit the game and return to the title screen

Behind the scenes:

This implementation of JezzBall squeezes every byte of space using some old-school tricks. The entire Python source code was minified and then packed into a binary blob with classic LZ77 compression enhanced by literal RLE grouping.

When the game launches, the code is decompressed on the fly and runs directly from memory. This unpacking step completes in just a few dozen milliseconds – it's a nifty way to make the game super compact. In this case, minifying and compressing significantly reduced the source code size, bringing it down to less than 9 KB (plus the decompression routine, which, after minifying, takes only 322 bytes).

A better compression ratio is possible, but it's important to consider the decompression time. The game needs to launch almost instantly, so a compromise was made between the size of the compressed source code and the game's startup time.

For reference, the original, unminified and uncompressed Python source code is included in the project archive.

And that's not all...

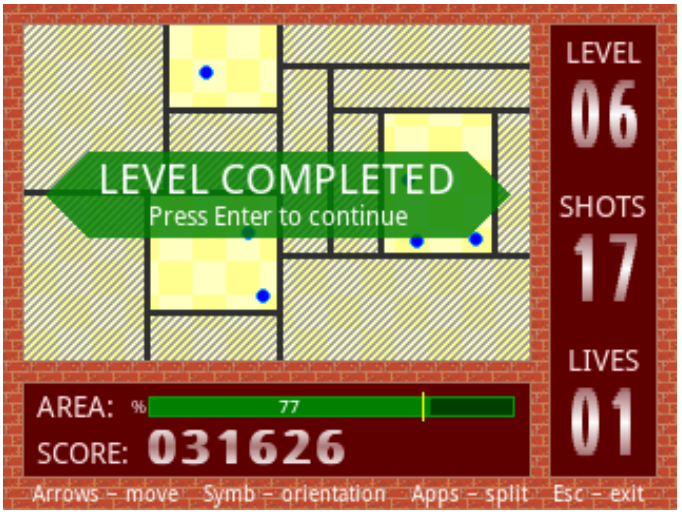
Wherever possible, assets were generated directly in the source code rather than stored as raw images, if doing so saved space. For example, all shaded digit fonts are drawn entirely in code instead of using bitmaps. As a result, the game includes only a single embedded bitmap – a tiny 37x17 pixel image taking up just 392 bytes. This compact sprite sheet contains graphics for the splitter (two horizontal and two vertical variants), the wall tile (which is tiled as the background), the ball, and a "+" symbol that appears next to the shot counter when it exceeds 99.

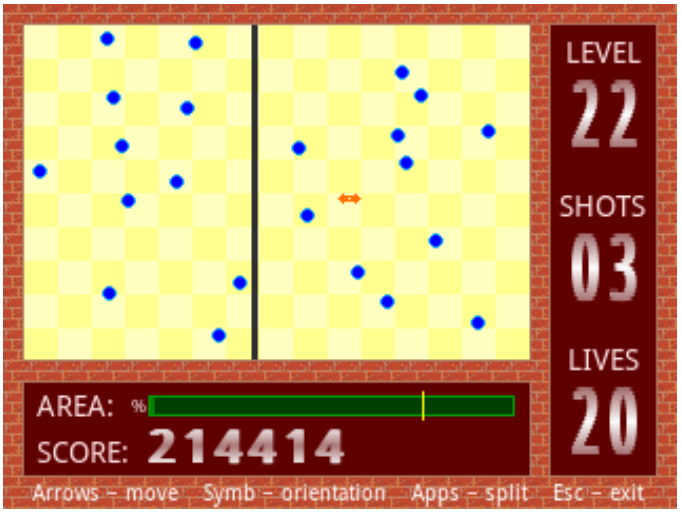
And one more trick...

The game keeps track of the high score, and even if you interrupt it at any moment using the [On] key, your score will still be saved. The program handles interruption events explicitly and ensures that the current high score is written before the game exits. This kind of handling is currently not possible in pure PPL.

And finally, an interesting fact: The ball collision detection and physics engine went through several rounds of performance optimization – with reasonably good results. The rendering algorithm also uses some clever techniques to squeeze out every frame: instead of redrawing the entire backbuffer on each frame, only the regions where balls appear are dynamically updated. Specifically, the rendering routine first erases the balls drawn in the previous frame, then renders the new ball positions onto the unchanged backbuffer. This incremental approach, combined with a few other low-level tricks, allows the game to run at a smooth 50 frames per second – even on the older HP Prime G1 hardware.

Screenshots:





Version history:

- **1.00 (2025-04-29):**
  - Initial release
- **1.01 (2025-04-30):**
  - Fixed an issue where the application failed to load under specific system settings (HSeparator).
  - Fixed the [Esc] key being ignored while extending the splitter.
- **1.02 (2025-05-04):**
  - Fixed missing labels, counters, and balls when the integer base was not set to hexadecimal in Home Settings before app start.
  - Reworked the ball collision logic to use integer arithmetic, improving physics accuracy and performance.
  - Reworked the application exit handling by replacing the try-except block with a context manager, including proper handling of user interruptions.
  - Refactored the class structure to improve separation of concerns.
- **1.10 (2025-05-07):**
  - Fixed an issue where letters on the title screen were incorrectly tilted along the arc when the angle unit was set to degrees before app start.
  - Implemented high score tracking, including level information, displayed on the title screen.
  - Improved the title screen layout for better readability.

- **1.20 (2025-05-12):**
  - Fixed the fill indicator bar that was visually stuck at 78% despite correct numeric value.
  - Fixed an issue where the high score level was not updating when the same points were achieved on a higher level.
  - Reworked the high score system to store data in a file instead of using AVars.
  - Reworked the collision detection logic for further performance improvements.
  - Improved the frame rate from 30 FPS to 50 FPS for smoother rendering.
  - Optimized attribute access by using local variable caching.
  - Minified and compressed the Python/PPL execution code inside the app using LZ77 algorithm, with on-the-fly decompression at runtime (G1: 150 ms, G2: 70 ms), reducing the app size.
- **1.21 (2025-05-17):**
  - Implemented high-score reset via the application-reset option.
  - Refactored the loader/decompressor to avoid retaining compressed and decompressed buffers in memory after decompression (only compiled bytecode remains).
  - Re-compressed the minified Python/PPL execution code using the LZ77 algorithm with literal-RLE grouping applied to improve compression ratio.
  - Extended the decompression logic to support literal-RLE decoding alongside LZ77.
  - Improved decompression performance (G1: 130 ms, G2: 54 ms).
- **1.22 (2025-05-19):**
  - Changed loader/decompressor to be imported directly (previously loaded via open → compile → exec).
- **1.23 (2025-05-20):**
  - Fixed typo in “LEVEL” (previously spelled “LIVEL”).