Video Game: The Assembler
File: TheAssembler

Lee Painton
Montclair State University
1 Normal Ave
Montclair, NJ 07043
paintonl1@mail.montclair.edu

David Heale
Montclair State University
1 Normal Ave
Montclair, NJ 07043
spitfire519x@gmail.com

Thomas Rademaker
Montclair State University
1 Normal Ave
Montclair, NJ 07043
pluginbaby94@gmail.com

CHI'13, April 27 – May 2, 2013, Paris, France.
ACM 978-1-XXXX-XXXX-X/XX/XX.

Abstract
In this document, the author describes the specifications of and design process behind the game titled, "The Assembler." This includes a brief discussion of the computer architecture concepts covered as well as references to related games.

Author Keywords
Minecraft; intrinsic learning; educational software

ACM Classification Keywords
Macro and assembly language; input/output circuits; simulation;

Introduction
Our game, *The Assembler*, started as a final project for a Computer Architecture course authors Heale, Painton and Rademaker were taking together. The idea was simple, take the major concepts taught in the course and design a game to teach and strengthen them. It is an example of an intrinsic learning (Ault et al., 2009) game where learning occurs as part of the process of meeting game objectives. To that end we are proud to present The Assembler to the CHI Student Game Competition.

ACM Copyrights & Permission Policy

Accepted extended abstracts and papers will be distributed in the Conference Publications. They will also be placed in the ACM Digital Library, where they will remain accessible to thousands of researchers and practitioners worldwide. To view ACM's copyright and permissions policy, see: http://www.acm.org/publications/policies/copyright_policy

Inspirations of Design

The Assembler is software designed to educate but it targets older students either in secondary school or university. When designing this game we took inspiration from games like Minecraft (www.minecraft.net) that present their players with lightly structured environments to play with. The educational value of these games is in their modeling of logical concepts in their own simulated environments.

As an example, Minecraft uses a basic set of on/off rules for a substance it calls 'redstone' which allows the p`layer to create complex logic gates and circuits. There is little documentation for redstone in Minecraft itself, but despite this the player community has produced a wealth of information about the logic circuits which can be created with redstone. This phenomenon inspired us to create a similar environment in our game. It is our hope that with the sandbox we have provided students can experiment with and develop an interest in the subject matter.

Background Narrative

The Assembler seeks to engage the player with a storyline and an objective of survival. Because the subject matter is technological we choose a science fiction setting. The player takes on the role of a janitor who is abandoned alone on a spaceship which is adrift in space. In this role he or she must repair the ship systems or be lost forever.

The choice of a janitor as the main character was to reflect a limited technical background in the player. Since the game is designed around a player learning as he goes we hope that she will sympathize with the main character and feel engaged in the knowledge he or she is learning.

Figure 1: The first game level in *The Assembler*. It depicts a system console that must be repaired by the player to survive.

The level we have provided for CHI presents the player with a problem solving environment. In this environment the player is called on to repair several logic circuits. They must then write assembly programs which interact with the circuits. All of these activities must be completed to fix the air conditioning system so

that the player does not suffer ill effects as the ship drifts closer to a star.

Further levels are planned which will call on the player to fix the ship's drive systems and write programs to calculate orbital trajectories. While we place some emphasis on realism in the design, our main goal was to provide consistency between the narrative and environment. While the programs and circuits the player will work with have no practical value outside the game world, the knowledge is portable and will aid him or her in further study of computer architecture.

Logical Circuits

One of the major concepts covered in *The Assembler* is combinatorial circuits composed of simple logic gates. The scope of the game includes AND, OR, NOT, XOR and Multiplexer gates (one and two bit). It calls on the player to troubleshoot multiple circuits which serve different critical functions on the ship.

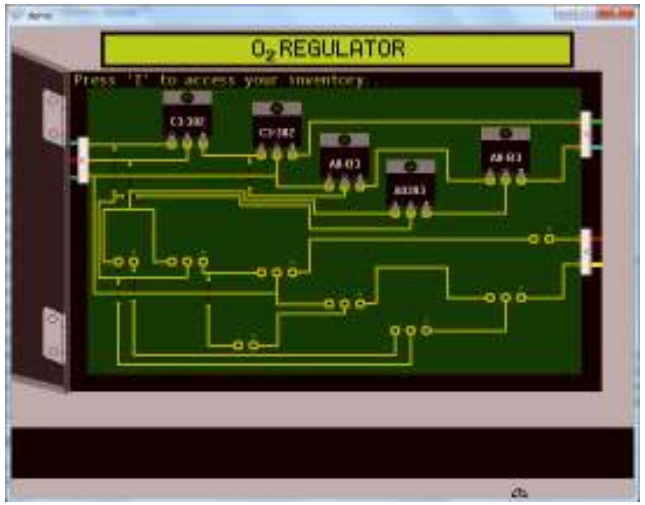

Figure 2: Figure depicts a panel with two logic circuits from the game. The microchips are logic gates. The completed circuit above is a full adder.

For the purposes of this submission we have included a full adder, full subtractor and a complex custom circuit which controls the ambient temperature of the simulated environment. The adder and subtractor are part of an ALU which is used by the assembler based systems to perform binary addition and subtraction. Thus the player will have direct control over the basic logical operations of the ship systems in the game. Developing and understanding of these circuits is critical to his or her success.

Assembly Language

It is perhaps no surprise that *The Assembler* incorporates assembly language into its learning objectives. The game seeks to assist the player in a basic understanding of how hardware and software

interact. Thus we have included a fully functional operating system to run programs which interact with logic circuits in the game. This operating system (called PAL) exists solely for the purposes of managing program files. It includes a basic text editor to write and edit programs and file system commands which manage the order programs are executed in. It also allows the player to check the status of the system's 100 memory registers.

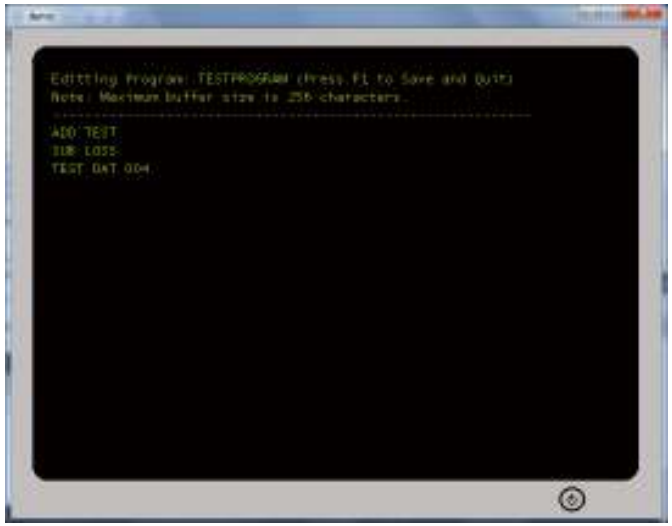

Figure 3: Figure depicts a program written on the in-game text editor provided for the player. This program is incomplete and would throw an error on execution.

The programs themselves are based on the assembly-like language developed for the Little Man Computer (http://www.atkinson.yorku.ca/~sychen/research/LMC/LMCHome.html) We have attempted to implement our LMC as faithfully as possible in order to give the player maximum flexibility in playing with their code. Our LMC simulator includes basic error handling to aid in troubleshooting. For the purposes of our game the player will need to write LMC programs to survive. We have reserved registers to either provide information input or accept output from programs. The player will use these reserved registers to interact with the system.

Our decision to use the LMC instead of a more practical language is that Little Man serves as a gateway language for assembly. The simplicity of its logic keeps programs simple while still communicating the mindset required to write low-level algorithms.

For more information on the Little Man Computer please see:
(http://en.wikipedia.org/wiki/Little_man_computer)

Special Thanks
We would like to thank Dr. Ursula Wolz for her encouragement and assistance in bringing this project to CHI. Her encouragement and advice have been invaluable.

Sources

1) [Ault et al. 2009] C. Ault, A. Warner-Ault, U. Wolz, T.M. Nakra (2009) "Kinesthetic Communication for Learning in Immersive Worlds." in Serious Game Design and Development: Technologies for Training and Learning, edited by Janice Cannon-Bowers and Clint Bowers. Hershey, PA: IGI Global, 2009. [Ault et al. 2009] C. Ault, A. Warner-Ault, U. Wolz, T.M. Nakra (2009) "Kinesthetic Communication for Learning in Immersive Worlds." in Serious Game Design and Development: Technologies for Training and Learning, edited by Janice Cannon-Bowers and Clint Bowers. Hershey, PA: IGI Global, 2009.