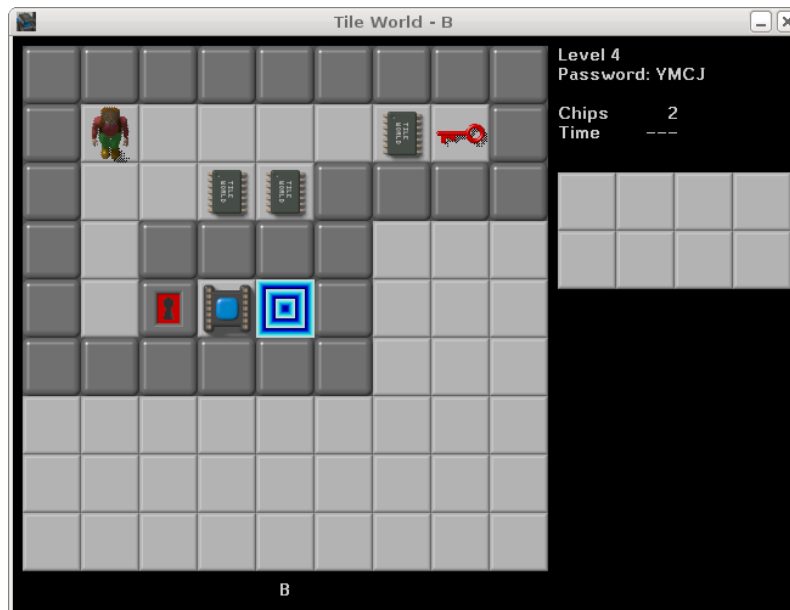


Due: 10:00 PM, Apr 2, 2009

Contents

1	Introduction	1
2	What To Do	2
3	Formulating Chip's Challenge as a logic problem	3
4	Support Code	4
5	Handing In	4



1 Introduction

Chip's Challenge is an old tile-based puzzle game primarily for Windows and DOS systems. The game stars Nerdy Chip McCallahan, a young man who dreams of becoming a member of the prestigious Bit Buster Club. In order to gain access to this club, he must pass a series of challenging tests to prove that he is, indeed, Bit Buster material. Each of these challenging tests corresponds to a level in the game.

In more gameplay-based terms, Nerdy Chip McCallahan's goal on each level is simple: reach a blue

portal to advance to the next level, where an even more difficult challenge awaits. However, every level's **portal** is blocked by a **chip socket** that cannot be passed through until all the **computer chips** on the level have been collected. Chip picks up any items he walks over. As he traverses the level collecting chips, Nerdy Chip McCallahan may come across different colored **locked doors** that obstruct his path; these can only be opened by finding a **key** of the corresponding color. Take a moment to identify the **blue portal**, **chip socket**, **computer chips**, **locked doors**, **keys**, and Nerdy Chip McCallahan in the given figure.

Your mission is to take a level as input and write the AI to help Nerdy Chip McCallahan complete that level. To do so, you will formulate a representation of the world in propositional logic, and will create a solver to determine how to reach the goal state in which Nerdy Chip McCallahan is located at the **blue portal**.

For this assignment, you need only worry about the level set the TAs have created, which only use a subset of the game's features. In this subset, the character's movements are subject to these constraints:

1. The player may only move onto a door square of a particular color if he has the properly colored key.
2. The player may only move onto the chip socket square if he has collected all of the computer chips in the level.

In each level, there is exactly one exit and no more than one key of each type.

2 What To Do

To begin, install the support code by running the install script:

```
/course/cs141/bin/cs141install chips
```

The support code will be installed into your `/course/cs141/projects/` directory. Your tasks will be the following:

1. First, formulate Chip's Challenge as a SAT problem, only taking into account walls and an exit.
2. Next, formulate Chip's Challenge as a SAT considering keys, doors, chips, chip sockets and an exit.
3. Implement the DPLL algorithm as a SAT solver for your Chip's Challenge SAT problem.

3 Formulating Chip's Challenge as a logic problem

The problem of determining whether it is possible to solve each level within a certain number of moves is equivalent to determining whether it is possible to fulfill a very large number of constraints. Some of these constraints may include:

- At time 0, Chip must be on the start square
- After his last move, Chip must be on the exit square
- If Chip moves south from the square at (2, 3) at time 2, he must be on the square at (2, 4) at time 3
- Chip can only be on one square at a time

These constraints can be expressed as statements in first order logic, using these functions:

- $chipOn(x, y, t)$ iff Chip is on (x, y) at time t
- $hasKey(c, t)$ iff Chip has the key of color c at time t
- $hasComputerChip(n, t)$ iff Chip has the nth computer chip at time t
- $move(x1, y1, x2, y2, t)$ iff Chip started moving from (x1, y1) to (x2, y2) at time t (and Chip arrived at (x2, y2) at time t + 1))

For instance, suppose that $chipOn(x, y, t)$ is true iff Chip is on square (x, y) at time t. Then if Chip is on a 2x2 grid, $chipOn(1, 1, 0) \rightarrow \neg chipOn(0, 0, 0) \wedge \neg chipOn(0, 1, 0) \wedge \neg chipOn(1, 0, 0)$.

Your first task is find a method to generate the first order logic statements needed to solve a level given the level map and the desired length of the solution.

These first order logic statements can be converted to variables and clauses in propositional logic. For example, one could state $chipOn(x, y, t)$ iff the propositional logic variable $chipOn_x_y_t$ is true. In a similar fashion, you can create a propositional logic variable $hasKey_c_t$ to represent whether or not Chip has the key of color c at time t.

Your next task will be to complete the support code and make a level solver using the SATPlan algorithm. SATPlan starts by searching for a solution of length 0. If it cannot find a solution of a certain length, it tries to find one that is 1 longer, up to a certain maximum length.

At each step, the algorithm generates a SAT instance as described above. When there is a valid solution, it extracts a solution from the variable assignments which make the instance satisfiable. After completing the first task, your program should be able to solve all the levels in the `mazes.dac` level set. When your planner is fully realized, it should be able to solve all the levels in `keysnchips.dac` as well. The TileWorld interface will allow you to choose a level set when you start the program.

4 Support Code

Much of the support code is for interfacing with TileWorld and you need not worry about it. The classes you will need to concern yourself with are as follows.

- The `Frontend` class : You do not need to modify this class, but this is where the main method for the project is found. Call the main method to run the project.

- The `ChipPlanner` class : This is where the bulk of your planning work will be done. See the comments on this class's methods to see what needs to be filled in.
- The `LevelMap` class and the `tiles` package : The `LevelMap` class, unsurprisingly, represents a map of a game level. It contains an array of `AbstractTiles`, the subclasses of which can be found in the `tiles` package. You do not need to modify any of these classes, but you will need to interact with them when creating your plan.
- The `sat` package : This package contains many classes which are useful for representing SAT problems. Chief among them is `SatInstance`, which represents a SAT problem in its entirety. A SAT problem is made up of several `Clauses` and `Variables`, and a solution to a SAT problem is represented by a Map of `Variables` to `Asgns`. Be sure to read over the comments in these classes so you can familiarize yourself with how they are used.
- `WalkSatSolver` and `DpllSatSolver` : These two classes can take a `SatInstance` and return a solution to it, the former using the WalkSAT algorithm and the latter using DPLL. The `WalkSatSolver` class is filled in for you, and may be useful when testing your planner. You must fill in the `DpllSatSolver` class as part of the project.

When the main routine in `Frontend.java` is run, `TileWorld` will be started. Select a level set using the arrow keys. Every time a new level is started, the frontend will request a solution to the level by calling `getPlan` in `ChipPlanner` with the current map. After your code returns a solution, you can use the frontend's buttons to execute it either step by step or all at once.

5 Handing In

When you have completed the project, please hand in the following:

1. All of your code (including our support code).
2. A README, including:
 - (a) Things that may help us read your code: design quirks, conventions, etc.
 - (b) Standard bug reporting.
 - (c) Any comments that might help us improve this project for future.

Hand in by typing `cs141_handin chips` in the shell from the directory containing your work.

Good luck!