Robotics Final Project Writeup
Kevin Scroggins
Team 2

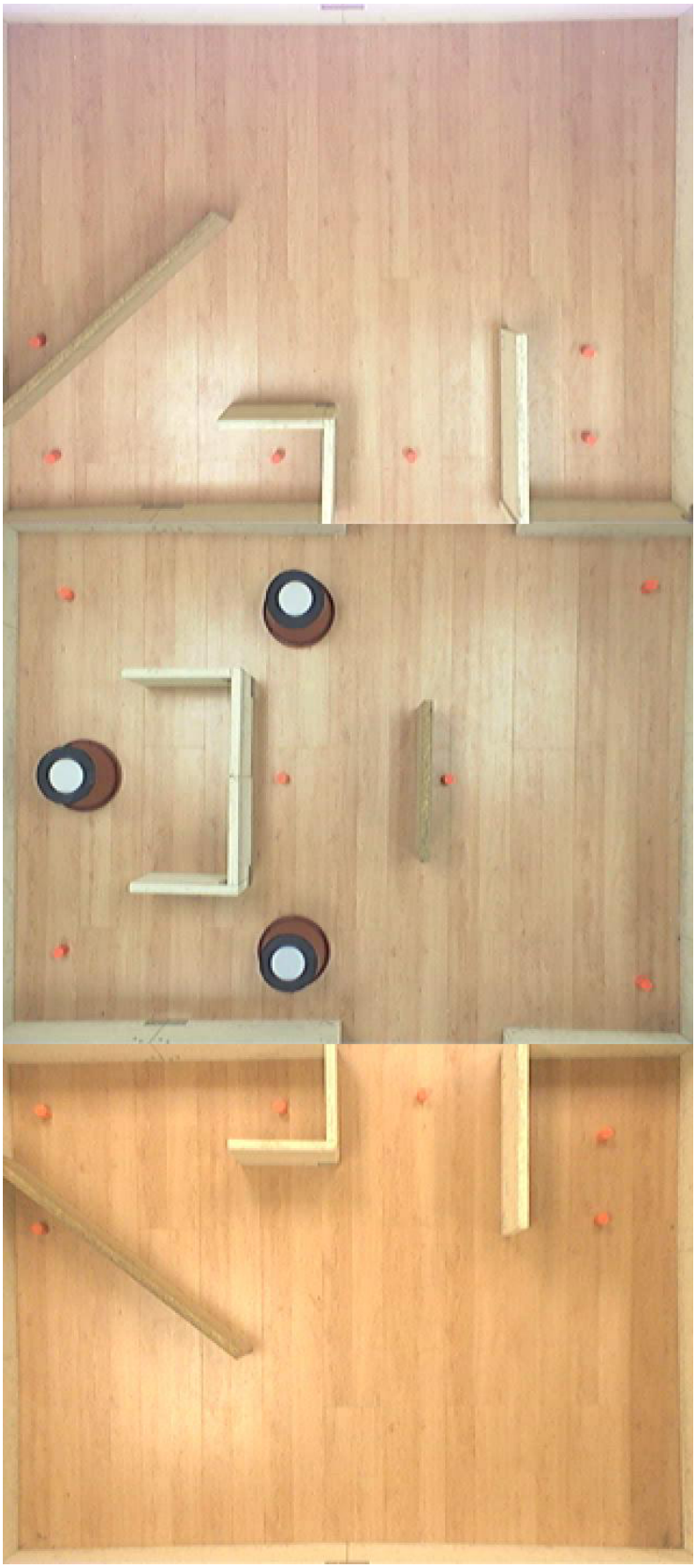
"The Competition"

The main goal for the compilation was for all the robots to work together through our 'Master Planner'. This planner would manage the robots locations across all three zones, displaying lots of debug information and synchronizing all relevant information to other planners via our central 'Master Server'. For the competition we used the backup code which used hard-coded points and behaviours, because the block deliver of our Master Planner was not functioning correctly at the time of the competition. This problem was resolved shortly after, however. We will be submitting a video of our actual code working instead of the demo video from the competition with the backup code.

Chris' goal was writing the back up code as a safety net, for if/when the Master Planner failed. This code was built for one robot that would be run in the center area. This robot would collect all the possible blocks using navigation and sweeping all the blocks to the drop-off zone. It also moves the left most pot to its' scoring zone in the middle of the run, and pushes another pot out of the way, getting some points. This code basically just used point-to-point navigation with some newly added functionality (i.e. **RAM THE POT!!!**, **NOS**, and sweeping of multiple blocks). It took quite long to fully get this back-up code working because we had to get the perfect point values, and those values required repetitive tweaking before arriving at the perfect value (for most consistent performance).

Kevin and Corey worked together as a team on the Master Planner and Master Plan (SPIN Code), both creating and debugging / testing it thoroughly. It is a very complex system with many useful features which are all outlined in the implementation details section.

Environment:



“The Team”

Roles: Because of the nature of our team arrangement, even though we had delegated primes, each of us helped out with each part of this project. Team members:

Kevin Scroggins
- Student ID: 100679071
- E-Mail: nitro404@hotmail.com

Corey Faibish
- Student ID: 100764177
- E-Mail: cfaibish@connect.carleton.ca

Chris Sullivan
- Student ID: 100744875
- E-Mail: yoshi_sully@hotmail.com

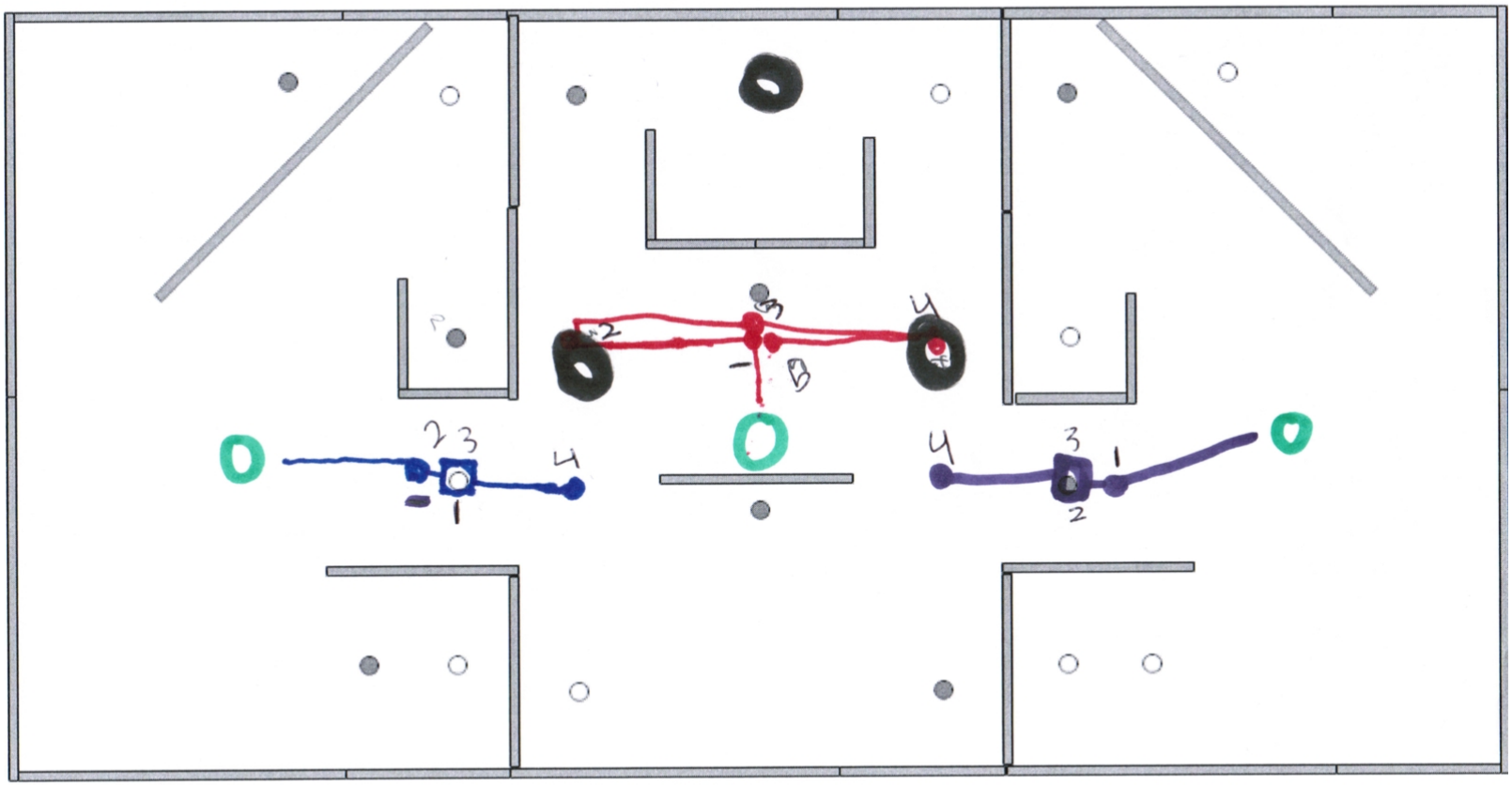
Robot Names:
Corey: Walter Chan
Chris: Star Warrior
Kevin: Pac-Man

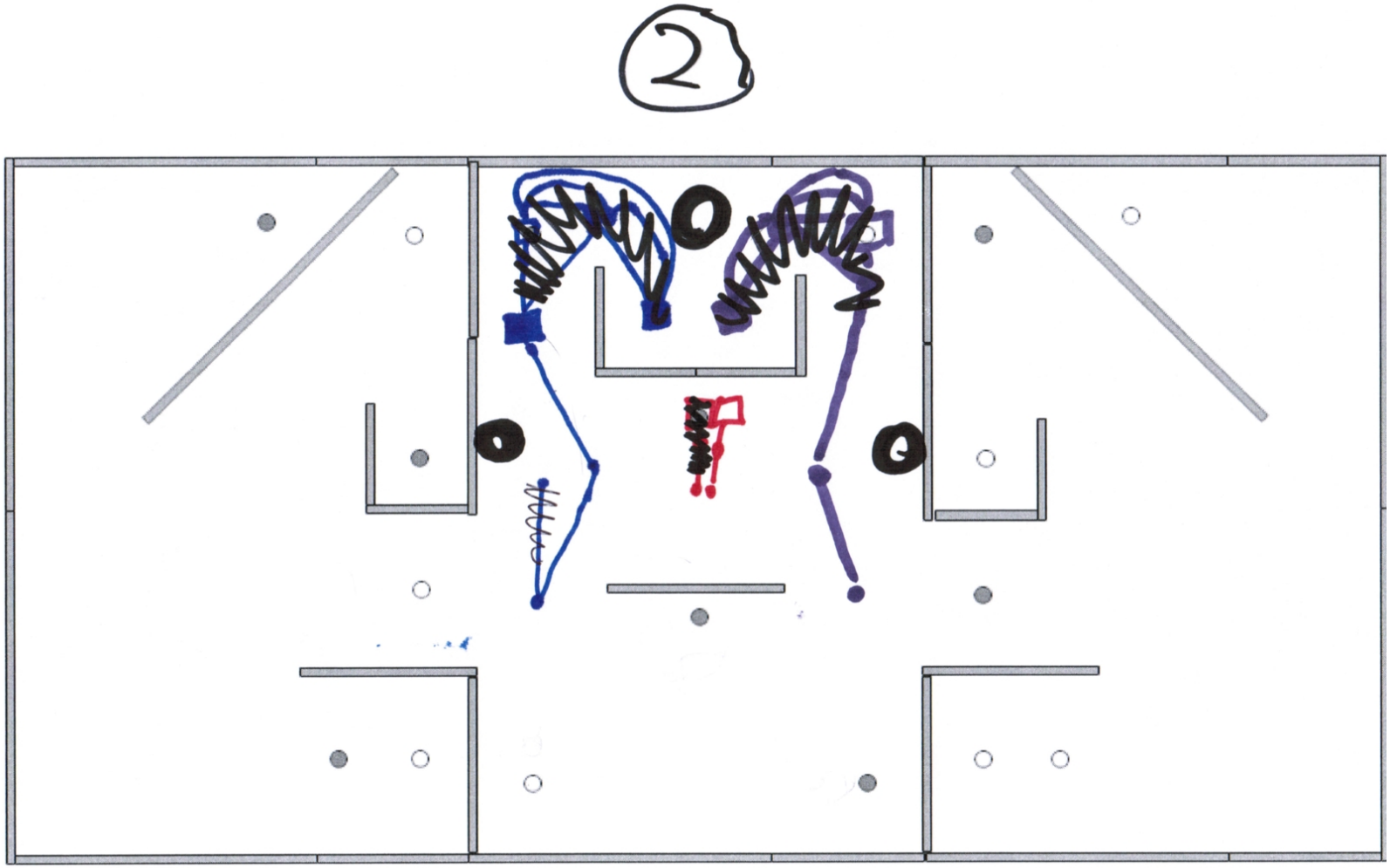
Who did what?

Corey	**Chris**	**Kevin**
GUIs for Master Planner	Back Up Code (BuC)	Master Planner
Master Server	(BuC) Navigation	Master Server
Master Plan (SPIN Code)	(BuC) Pot Ramming	Master Plan (SPIN Code)
FlowerPotPush	(BuC) Block Sweeping	Display GUI Window
FlowerPotSearch	(BuC) Block Delivery	RobotSystem
TaskManager	(BuC) Mini Master Planner	BlockSystem
Path Plotting	(BuC) SPIN Code	PotSystem
Navigation		PathSystem
BlockPickup		TaskManager
BlockDeliver		Navigation
		BlockPickup
		BlockDeliver
		Networking
		Serialization System
		Webcam Interface

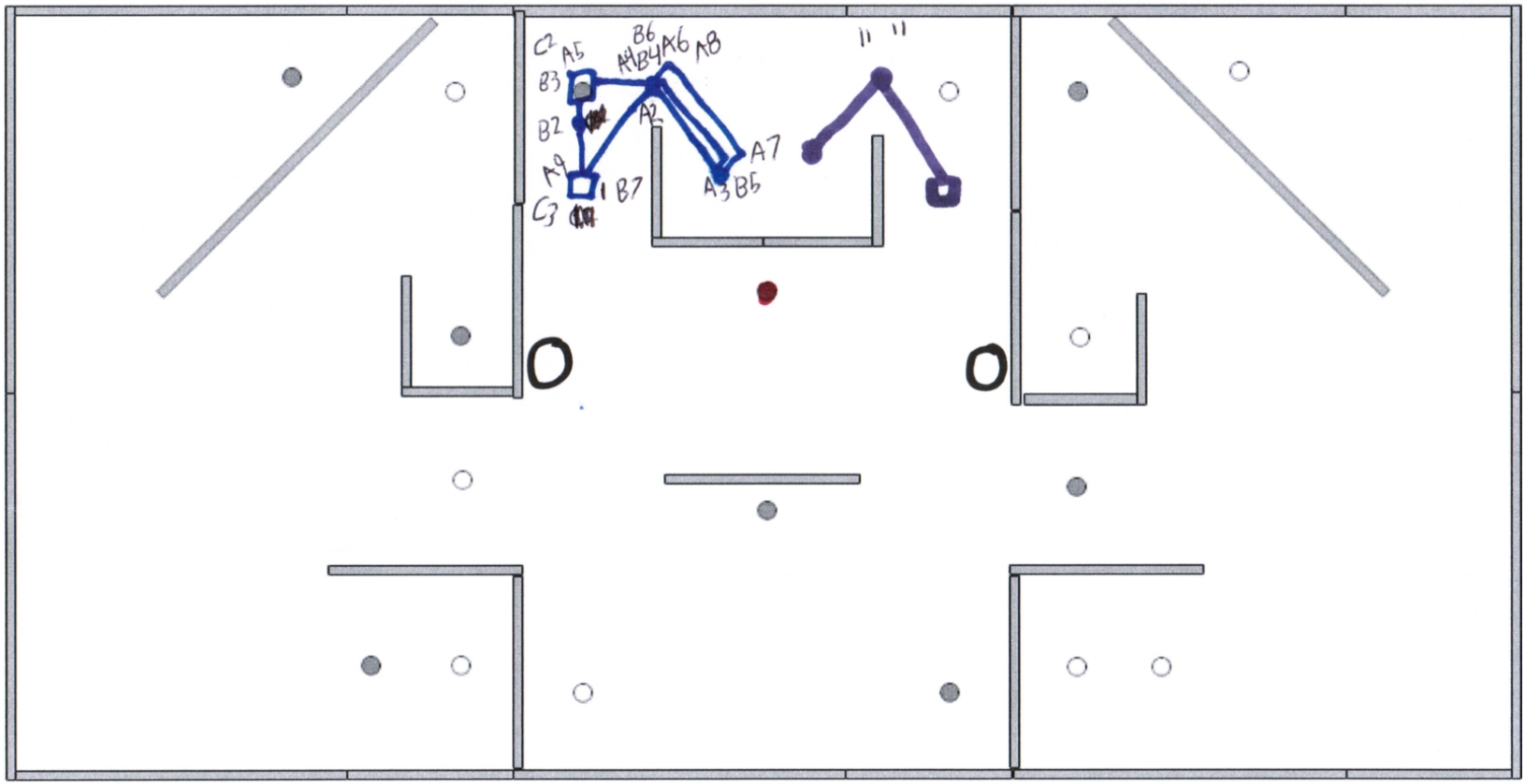
"My Approach"

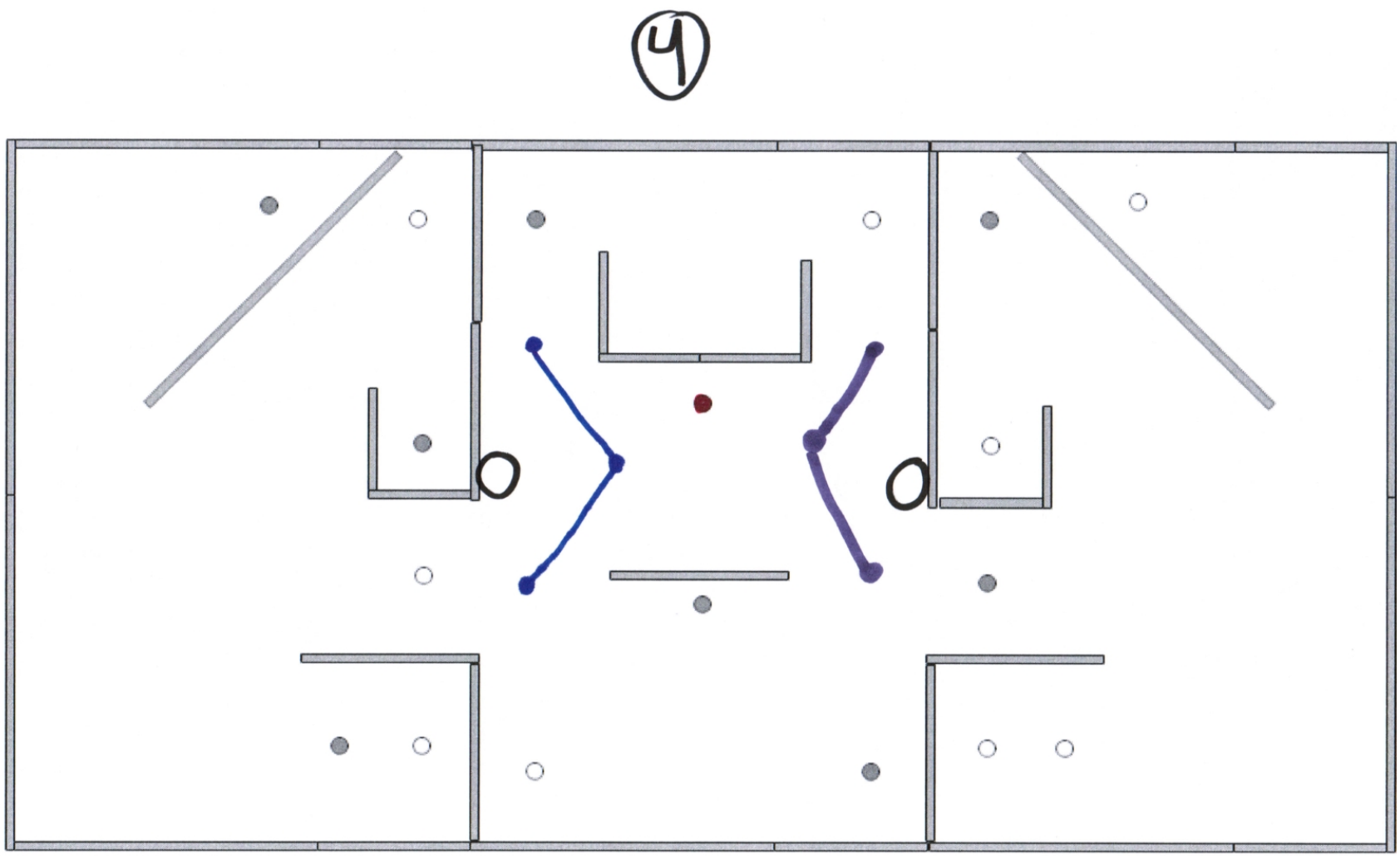
For our approach, we outlined a series of synchronized tasks via "flow diagrams" (essentially print-offs of the environment which we traced out objective paths on). Our implementation consists of a total of 13 tasks, each task containing a set of objectives (or in some cases, no objectives at all). We used squares to designate choice objectives, lines to designate paths and points to designate positions along our path(s). They are not overly detailed, as they were only used as a reference to help us plan our tasks for completing the objectives of the project. Each task and objective is numbered, however objectives do occasionally contain branching points where choices after the branch point are indicated by a different prefix letter. The following 13 pages are scans of our original "flow diagrams":


2
3
4
1
2
3
4
B
4
3
1
2

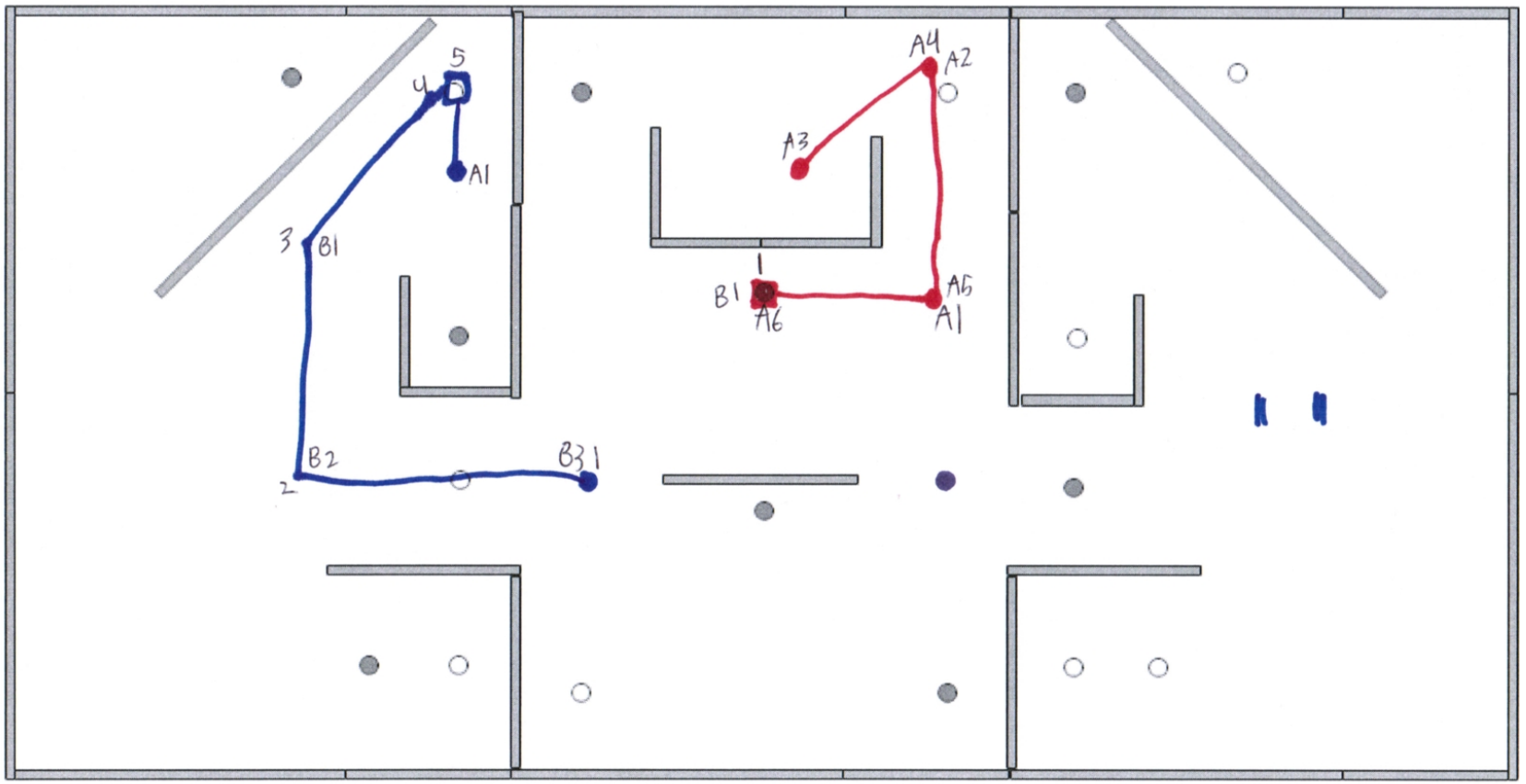

2

③

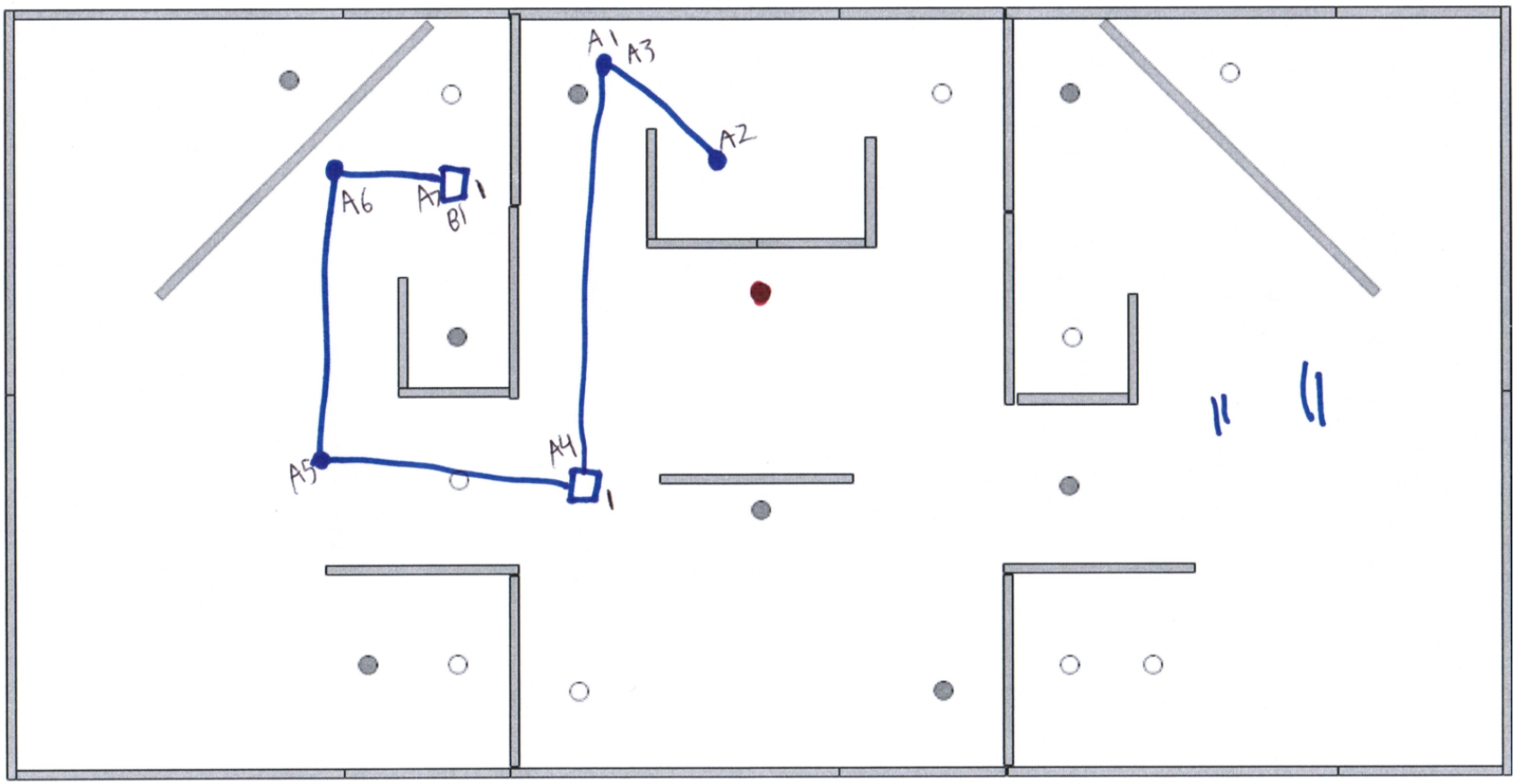



4

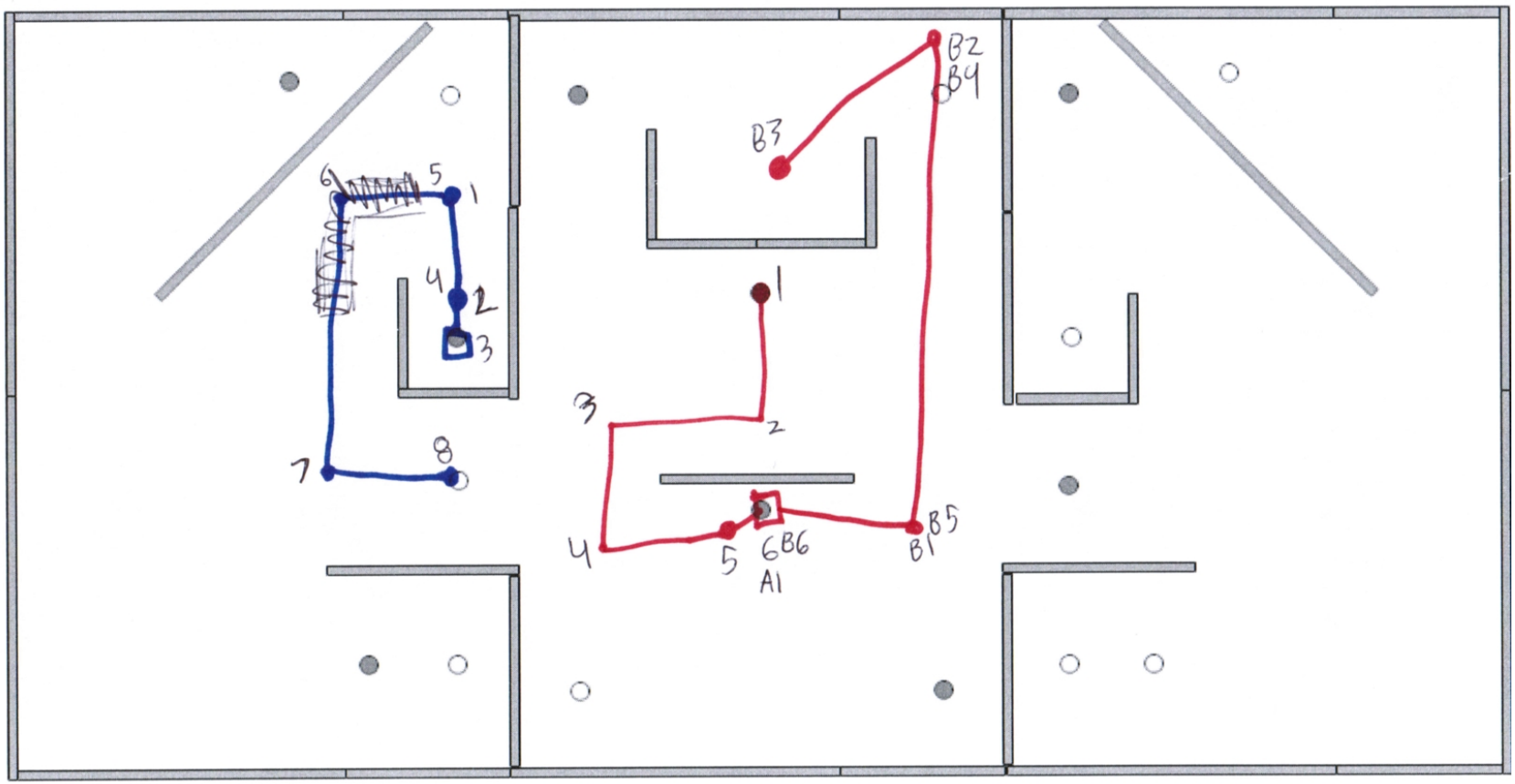
⑤



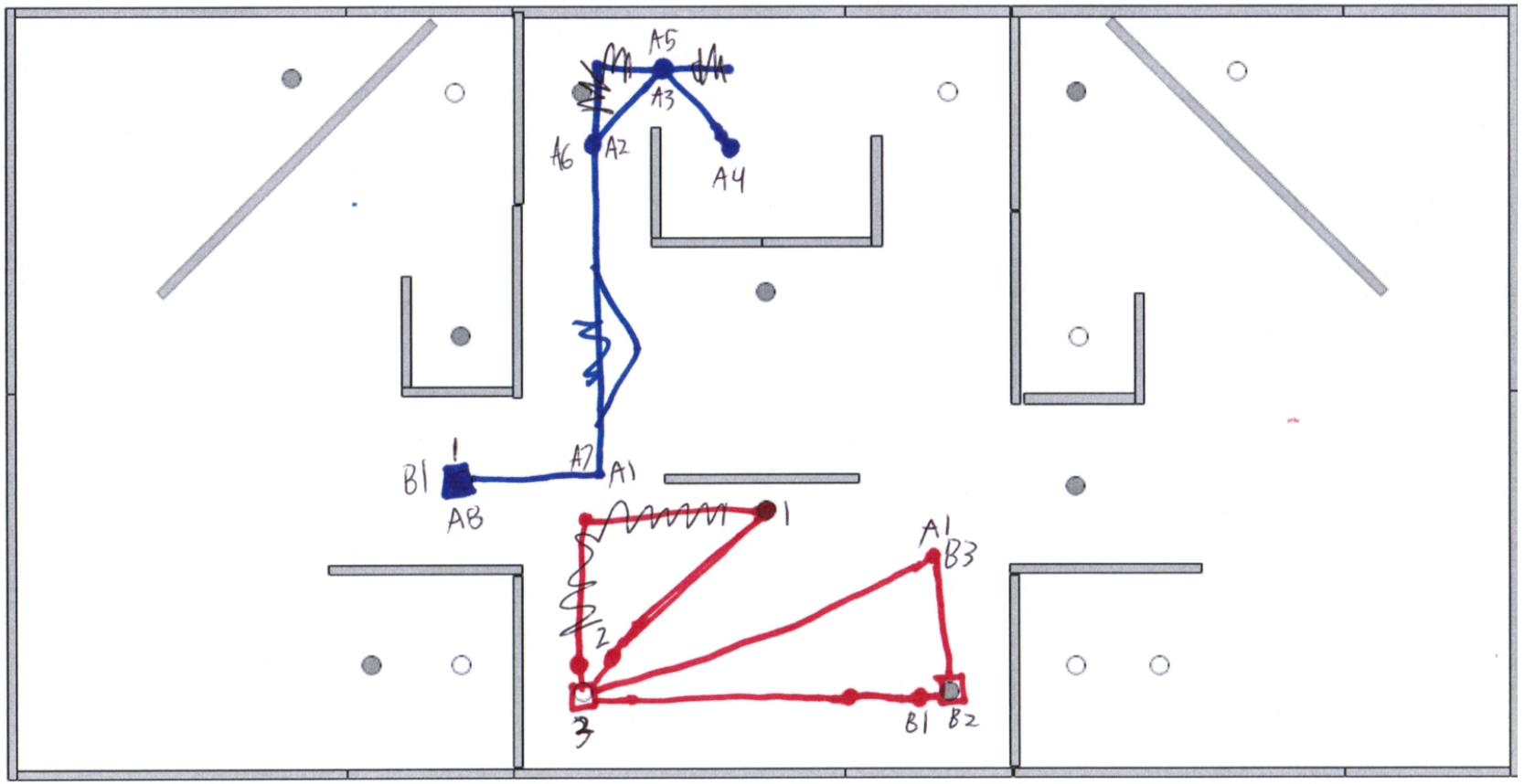
6



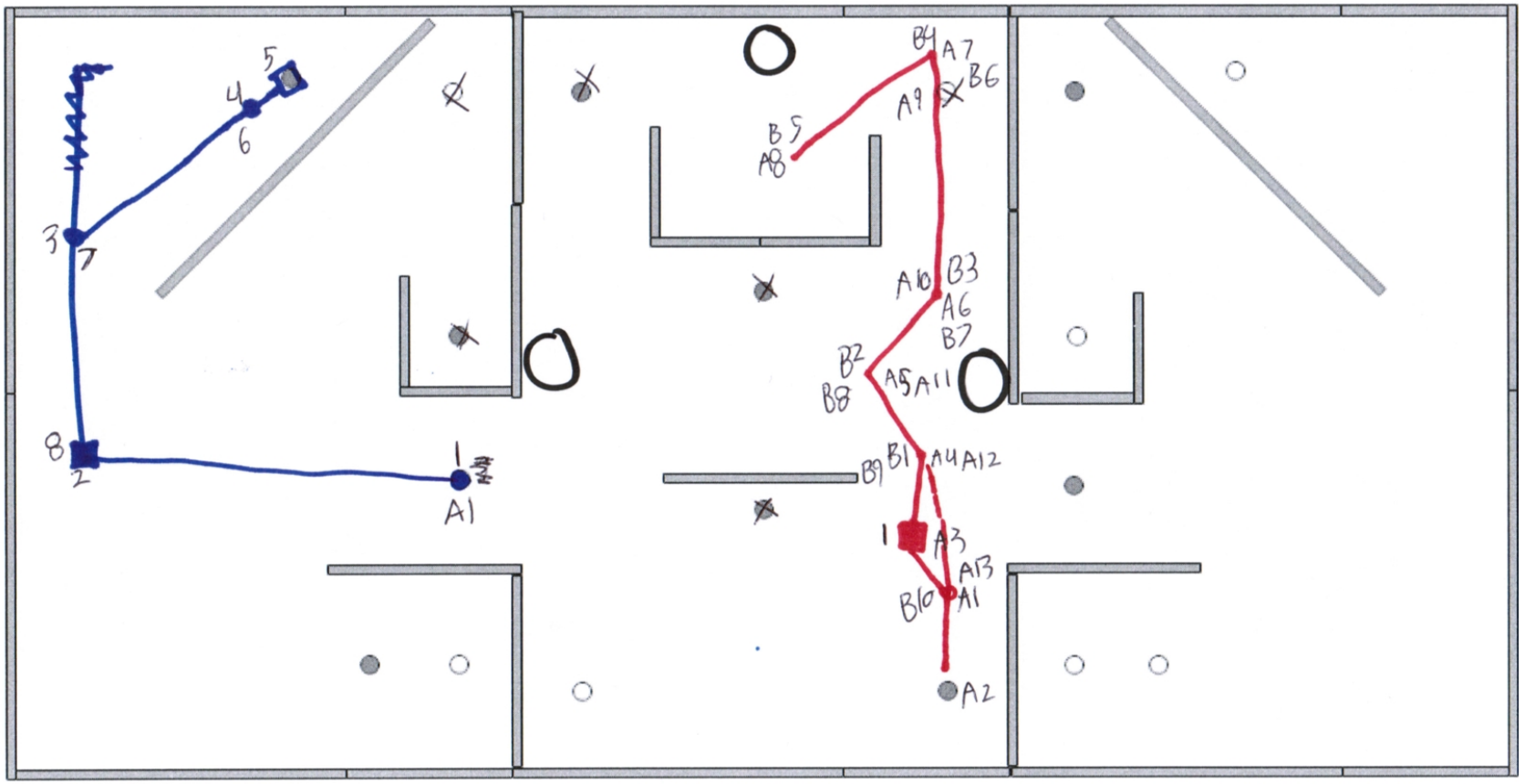
7



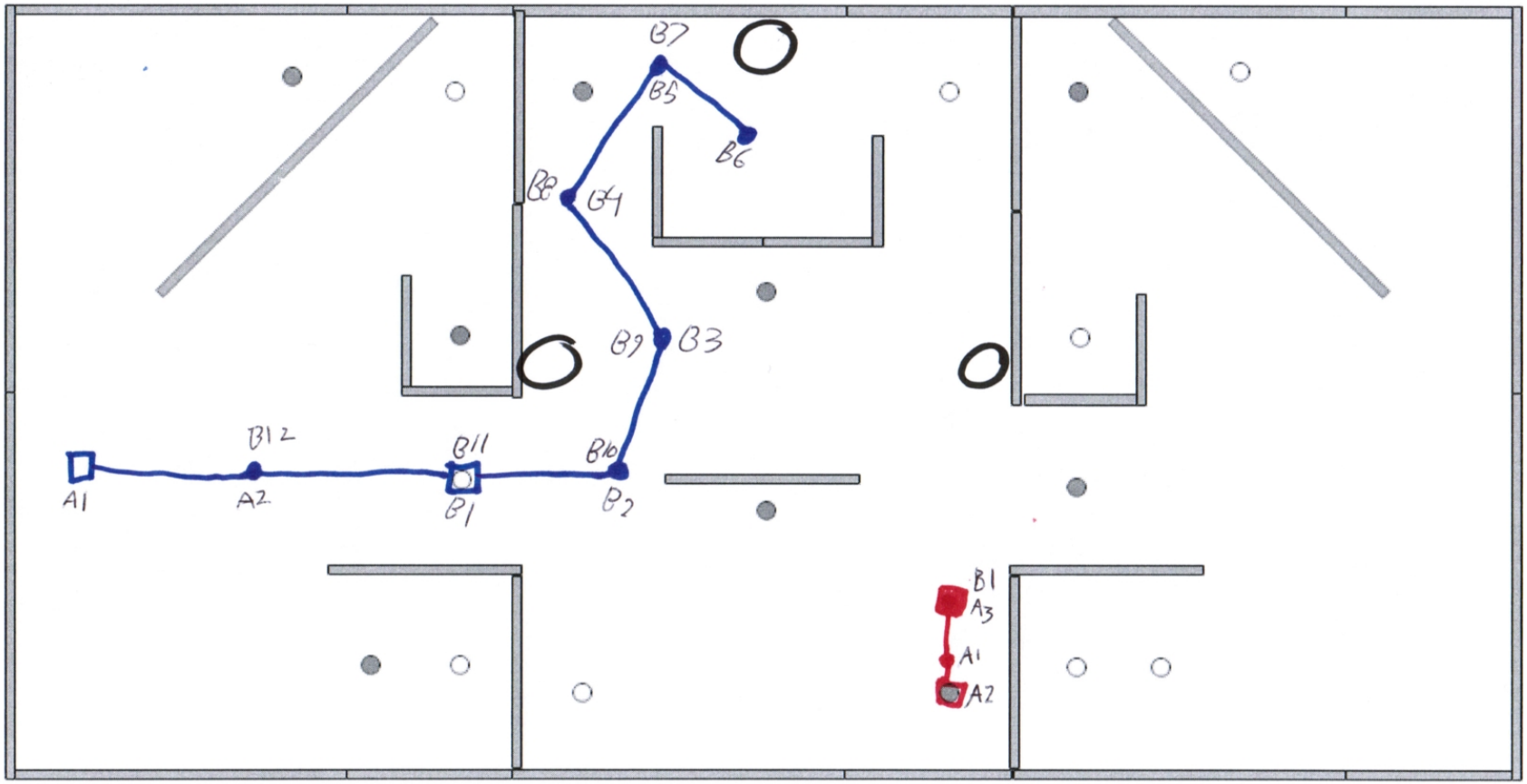
8



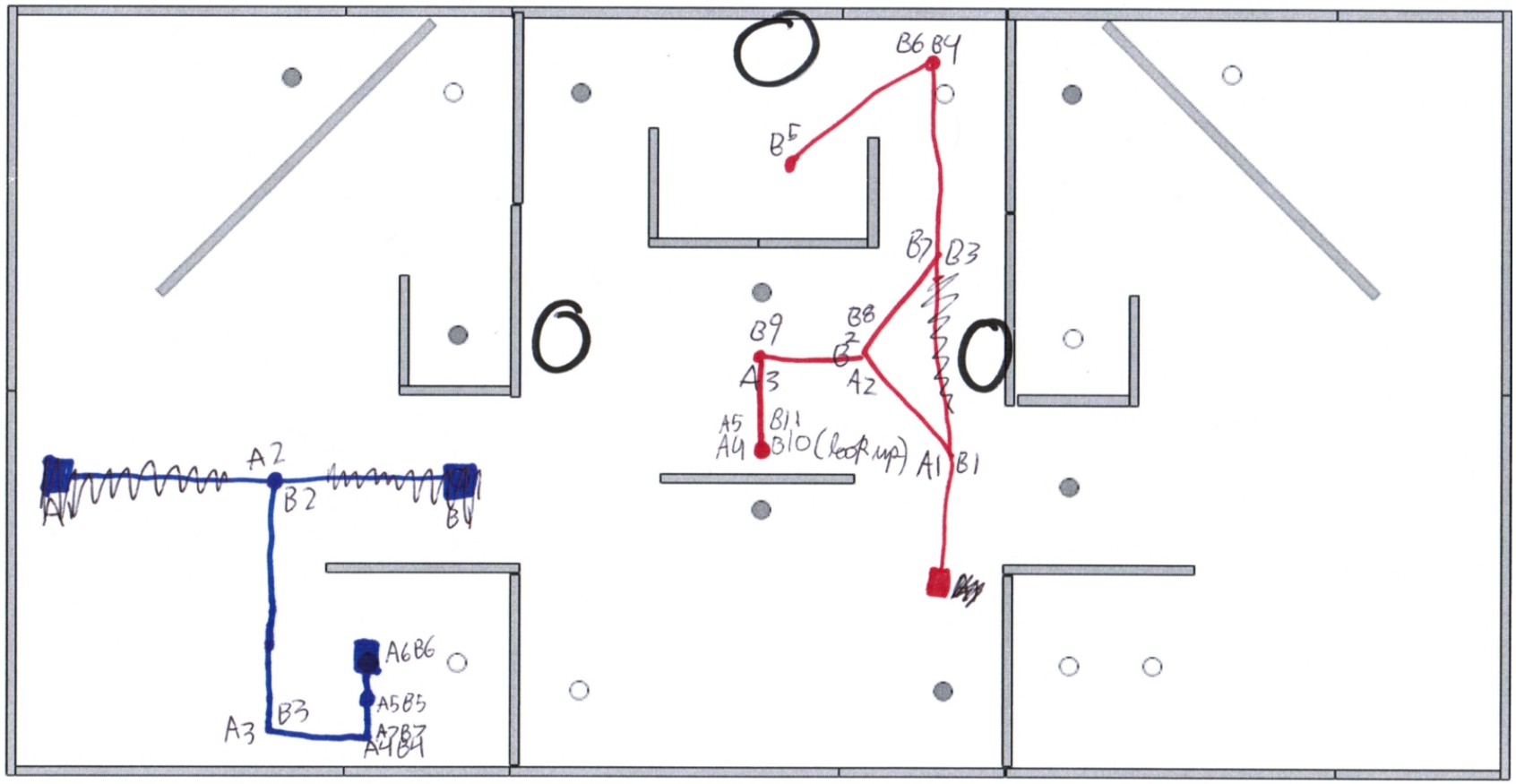
9


5
4
6
3
7
8
2
1
A1
B4
A7
B6
A9
B5
A8
A10
B3
A6
B7
B2
B8
A5
A11
B9
B1
A4
A12
1
A3
A13
B10
A1
A2

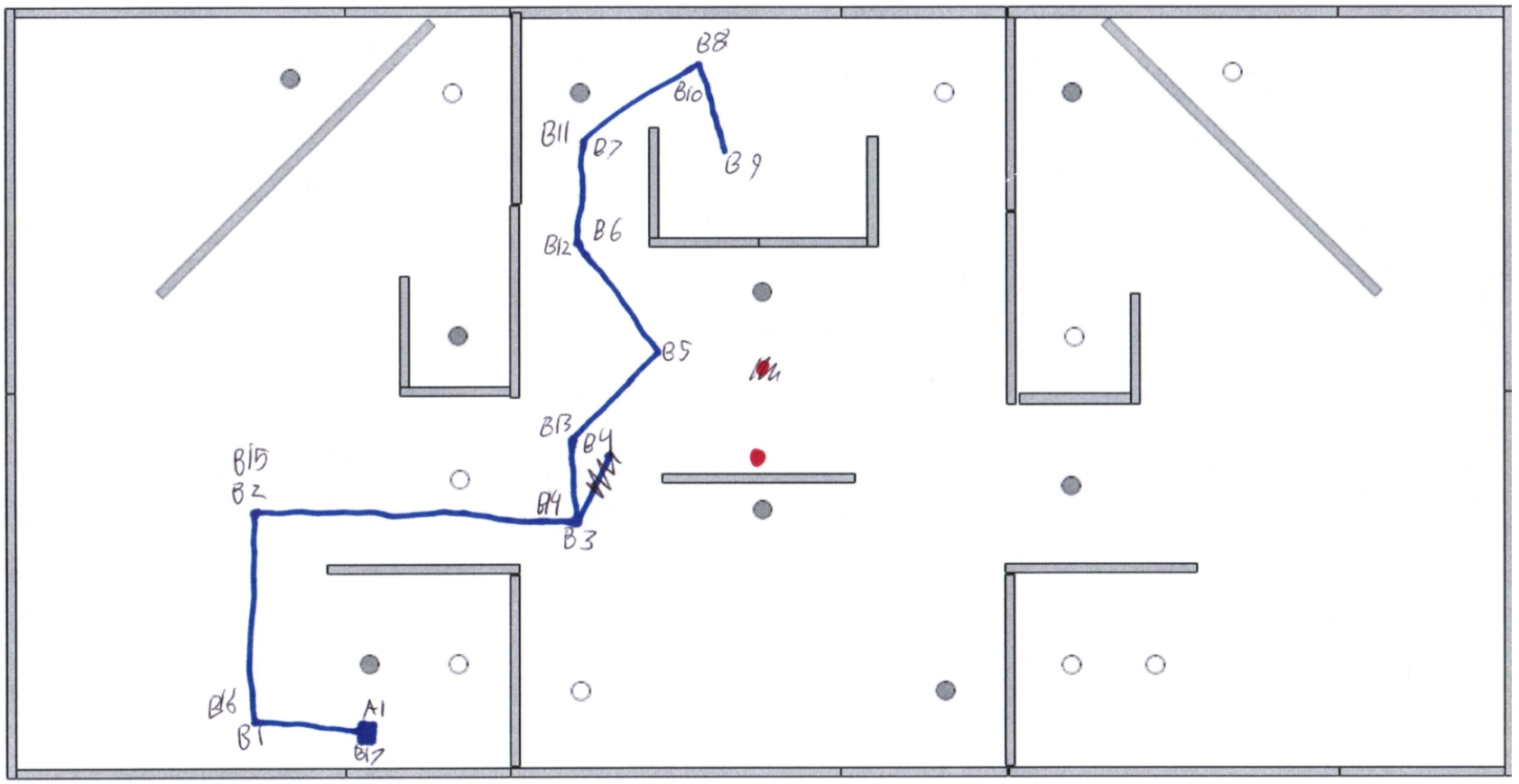
10



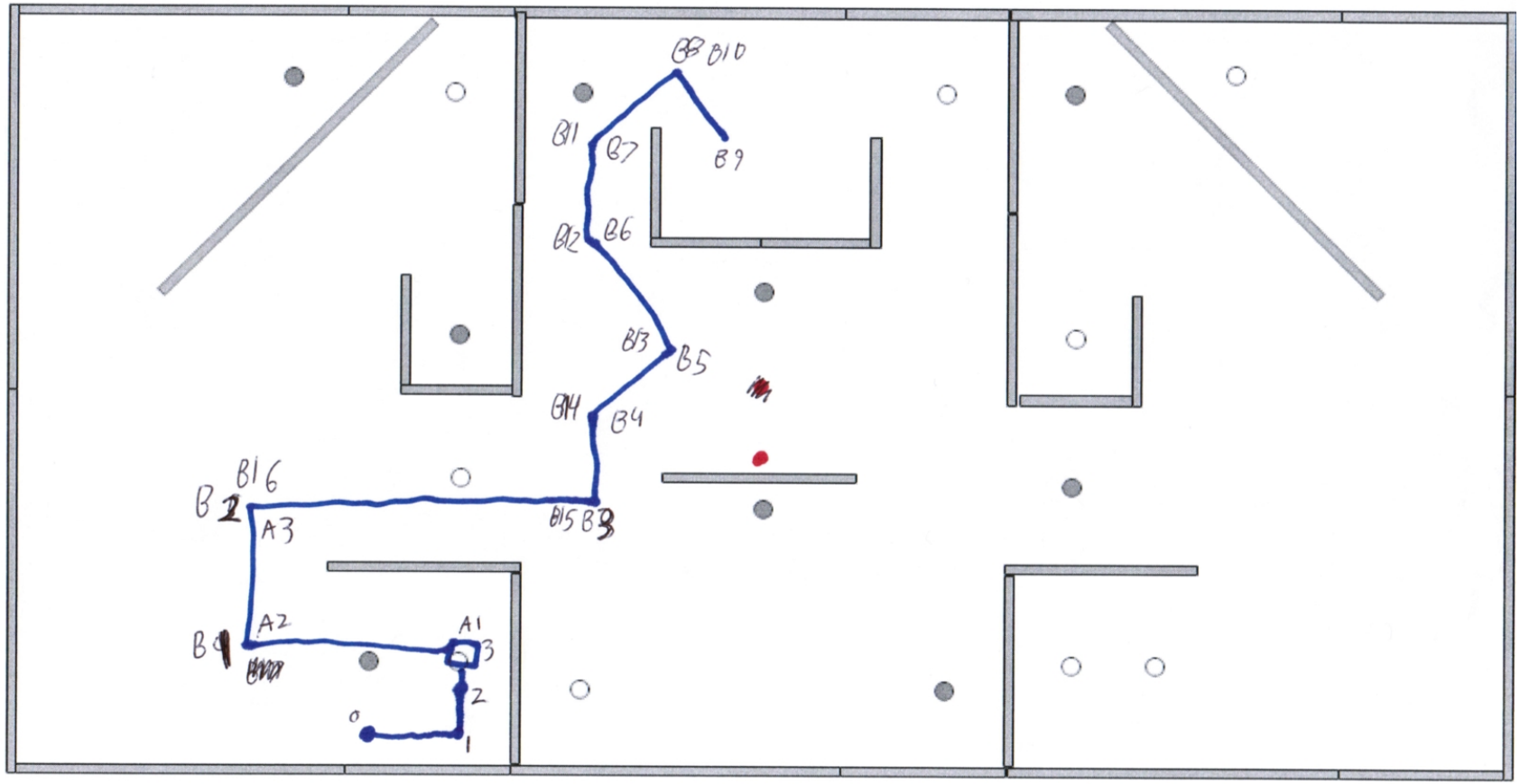
11


B6 B4
B5
B7 B3
B9
B8
A3
A2
A5
B11
A4
B10 (loop up)
A1 B1
A2
B2
A6 B6
A5 B5
B3
A3
A7 B7
A4 B4

12



13



“Implementation Details”

Our implementation for the project involved a very high level plan using point-to-point navigation, robot synchronization and simple instructions being sent to the robots (ie. MoveForward, TurnLeft, TurnRightSlowly, PickUpBlock, etc.). Instructions are sent from objective objects which are ultimately managed through the task management system. Tasks can be edited using our task editing window, and are automatically synchronized with a task list file, for portability. Once a robot finishes a set of objectives (a task), it waits for all other robots to finish their current task before continuing. It is assumed that tasks will be set up in such a way that they will prevent robots from colliding with each other (as we have no collision detection). Using a high-level plan such as this allowed us to avoid unforeseen issues like what would happen if two robots got too close to each other, or if wall following did not run as expected, for example. Our implementation ultimately contains 3 major systems: MasterServer (JAVA), MasterPlanner (JAVA) and MasterPlan (SPIN).

MasterServer - A central signal forwarding server which allows for reliable planner-to-planner communications. This means that planners can keep in synchronization with each other by forwarding any state changes, position changes or whatever other relevant signals there may be to all other planners through the master server. The master server has a few features, such as its own console for outputting signals being forwarded through it, as well as any other debug information. During the simulation, it generally remains idle and requires no user intervention (aside from assigning tracker numbers to clients when they first connect). We have designed our own protocol system for the purposes of this project which is shared and utilized by both the planner and server.

MasterPlannner - The main focus of our implementation. The planner contains a very nice graphical user interface to allow for easy visual debugging and monitoring of simulations, simple planning of objectives / tasks for each robot, and so on. There are two main graphical user interface windows: a “display window” and a “planner window”.

The “display window” shows all 3 tracker “zones” as background images along with circles which represent robots, blocks, pots and drop off locations, as well as segmented lines to represent the paths the robots will take when completing their tasks / objectives. The display window also contains functionality for dynamically loading and synchronizing updated tracker images between all 3 planners, but since we ran into some issues transferring the image data over the network, this feature is unavailable. The window is also highly interactive, and all pots, blocks and robots

can be re-positioned as needed. Paths can also be edited, created and deleted as needed (there is support for multiple paths, for more flexibility).

The “planner window” is mostly for textual debugging and changing settings. The main window shows information regarding robots, blocks and pots and generally indicates their positions / poses, states and any other relevant information. It also shows how many pots and blocks have been delivered and how many tasks each robot / all robots have completed. There is even a timer at the top of the window to keep track of how much time has elapsed since the simulation started. And lastly, there is a small console window at the bottom to monitor signals being transferred over the network. There is also a separate task editing window which can be accessed from the edit menu, which allows the user to plan tasks for each robot, and the objectives associated with each tasks. It is fairly intuitive and easy to use (although a little cryptic when trying to edit already-created tasks).

The simulation does not start until after you start the robots and the GUI windows load. Each planner must connect to the server, and once all planners are connected one of the users must start the simulation, then the robots will begin fulfilling their designated tasks / objectives. While the simulation is running, you will be able to see robots moving around, the directions they are facing as well as lines pointing towards their current destination (if any). If a robot picks up a block, you will be able to see the block moving with the robot on the “display window” and if a block is not found, it will be greyed out. This makes for really nice visual debugging / monitoring of our system / simulation as it progresses.

Our planner consists of a number of modular systems that make our high-level approach possible:

- **RobotSystem** (Robot, RobotState, RobotInstruction, RobotResponse, RobotPosition):
  Manages all of the robots in the simulation, including their states and poses. Contains sensitivity values for point-to-point navigation and handles responses from the robot when it is looking for / dropping off blocks. Keeps robots synchronized with all other planners as well as synchronizing their initial positions with the settings file.
- **BlockSystem** (Block, BlockState, DropOffLocation, DropOffLocationState):
  Manages all of the blocks / drop off locations in the simulation, including their states and positions as well as keeping them synchronized with all other planners. Keeps initial block and drop off locations synchronized with the settings file.
- **PotSystem** (Pot, PotState):
  Manages all of the pots, including their states and positions, as well as keeping them synchronized with all other planners. Keeps initial pot

positions synchronized with the settings file.

- **PathSystem** (Vertex, Edge, Graph, Path):
  Manages all of the paths used in the simulation for designating paths for the robots to follow. Keeps paths created by the user synchronized with the path data file.
- **TaskSystem** (Objective, ObjectiveMoveToPosition, ObjectiveBackUpToPosition, ObjectiveLookAtPosition, ObjectivePickUpBlock, ObjectiveDropOffBlock, ObjectiveSkipTo, ObjectiveChoiceBlock, ObjectiveLast, ObjectiveType, ObjectiveState, Task, TaskList, TaskState, NextTaskType):
  The central controller of the simulation. When the simulation is active, this system will be automatically updated each time a robot pose is received for the robot associated with this planner. Each time the system updates, an command will be sent from the current objective of the current task to the robot, instructing the robot on what it should do. If the robot has finished all of its objectives within the current task it will remain idle, waiting for the other robots to finish before continuing.
- **Client** (ClientThread, ServerDisconnectHandler, ServerInputSignalQueue, ServerOutputSignalQueue):
  The code for handling networking and signal i/o with the master server. Updates the system as appropriate with signals received from other planners.
- **Signal**:
  The signals transmitted over the network during planner-to-planner communications.
- **Planner** (MasterPlanner, SystemManager):
  Contains the central system which manages instances of all our systems, as well as communication between the planner and our systems. The actual planner class file is essentially a "dummy" planner which invokes the system manager and forwards any data it receives to the system manager.
- **Imaging** (Webcam):
  Contains our webcam abstraction which allows us to interface with the computer's webcam.
- **Shared**:
  Contains code which is shared between the client and the server. This includes the system console, updatable interface and automatic updater, positions and our serialization system (bytestream), etc.
- **GUI** (DisplayWindow, DisplayPanel, PlannerWindow, TaskEditorWindow, EditMode):
  Contains all of the GUI code used in our implementation.

MasterPlan - The code which directly controls the robot. This portion of our implementation is fairly simple and generally serves as a command interpreter for signals sent from the planner (objectives), ie. MOVE_FORWARD = 0, PICK_UP = 12, etc. Robot movement during point to point navigation also has features for turning / moving slowly, as well as arcing instead of turning to help speed up the completion of objectives / tasks (ie. so the robot does not overshoot / overturn past its destination, and instead of turning if it is no longer moving straight it arcs back until it is - which is faster than turning). The largest portion of code on this part of our implementation is the block pickup / drop off code which works independently (ie. only requires a single command to search for and pick up a block / drop off a block and back up a little).

The following are lists of the network signals and robot instructions used by our system:

Network Signals

0. Ping
1. Pong
2. StartSimulation
3. BlockStateChange
4. RobotStateChange
5. PotStateChange
6. TaskStarted
7. TaskCompleted
8. UpdateBlockPosition
9. UpdatePotPosition
10. UpdateActualRobotPosition
11. UpdateEstimatedRobotPosition
12. RequestTrackerImage
13. ReplyTrackerImage
14. BroadcastTrackerImage
15. ReceiveTrackerNumber

Robot Instructions

0. Null
1. Stop
2. MoveForward
3. MoveForwardSlowly

4. BackUp
5. BackUpSlowly
6. TurnLeft
7. TurnRight
8. TurnLeftSlowly
9. TurnRightSlowly
10. ArcLeft
11. ArcRight
12. PickUp
13. DropOff
14. OpenGrippers
15. CloseGrippers
16. Finished

"Problems Encountered"

1. Robot Tracker send the wrong position information (when you were close to certain walls) and the angle information was never right. This made navigation hard because the robot would spend 3 times as long trying to find the angle it should be at, but it always over shoots it due to the tracker. As for the -1 position in the BuC the robot would spend about a minute in the corner trying get its position again after it had a pot in gripper.
2. Robots stopping; "dying".Even tho they are reserving data and its displaying on the RBC the robot servos do not move.
3. gripper do not close all the way
4. Walls and floors not being painted a neutral colour (ie. black / white) results in robot occasionally picking them up as a block
5. Computers / webcams / network / software highly unreliable in all respects
6. Network highly unreliable, no poses received for up to 10-15 seconds at some points, robots relying on point to point navigation and would result in them running into walls or continually spinning in circles / back and forth since the tracking system does not work as expected
7. All network traffic has high potential to get corrupted and typically does (due to high byte conversion issue), includes:
    a. Desired paths from tracker
    b. Poses from tracker
    c. Robot to planner communication
    d. Planner to robot communication
    e. Tracker to tracker to communication
8. Computers are not grounded, resulting in constant shocking and frying of computer USB ports and Robot's chips (amongst other things) .
9. Webcams have a tendency to "shake/vibrate" while using

10. Dead zones using webcam tracker where it is impossible to see the robot, resulting in the robot getting stuck when using point to point navigation
11. Webcams settings constantly reset and are sometimes not settable until after the tracker is started
12. Webcams occasionally fail and need to be unplugged / plugged back in which only sometimes works and can result in the computer blue-screening
13. A typical 3 hour lab results in at least 5-10 computer reboots, wasting most of our lab time
14. Computers not properly set up - inconsistent settings and software on each
    a. Webcam on 3rd station never works properly ever
    b. Network not properly configured on 3rd station, resulted in any networked assignments / projects being literally unable to run
    c. Second station had far more software running as background processes causing much more lag on the computer and resulting in trackers running even more poorly
    d. Mice never work on the first station, had to bring own mice to school
15. Network appears to become more congested at random points causing any networked code to not work (at best)
16. RBC program is terrible
    a. Crashes constantly (ie. Actual unhandled exceptions)
    b. Unintuitive user interface (ie. Unable to start robot on some occasions, annoying to use)
    c. Planner loading is terrible (ie. Class naming conflicts due to inconsistent software, tedious task to load planners, packages not supported, etc.)
    d. Planners / planner loading contain cryptic error messages (ie. no stack traces to tell you what went wrong - just a simple message saying some function could not be called)
    e. Inconsistent versions between jar and version on computers and version on website
    f. Copying data out of the log windows in the RBC is very unintuitive at best
    g. Contains hard-coded paths, non-portable
17. RobotTracker program is even worse
    a. Does not remember settings from last run (or at least doesn't update the GUI properly, causing many issues)
    b. User interface is not friendly (ie. Issues when saving video files)
    c. Webcam image would simply freeze on occasion for no reason, usually during the middle of a simulation resulting in us having to painfully restart time and time again, usually once it is at least 20 minutes in
    d. Pre-plotted paths even have a tendency to get corrupted (likely due to high-byte issue)

    e. Estimated path still hard-coded in "fixed version"
    f. Unable to plot desired paths in "fixed version"
    g. Tracker version on website versus version on each individual computer were each different and were missing different features / had different things wrong with them (huge inconsistencies)
    h. Path planning needs improvement - can only have one path, on one zone, cannot move points or delete specific points, etc.
    i. Code for syncing poses between trackers needs improvement badly - would result in no poses being received for up to 10 seconds in some cases
    j. Contains hard-coded paths, non-portable
18. Propeller tool is awful
    a. Constantly crashes while loading code and the only way to fix is to reboot the computer (and it tries to disable the "failed" com port - likely due to the computer not being properly grounded)
    b. Even after the program has been closed it is still running in the background and using 50% of the CPU, resulting in more tracker lag and terrible results
    c. IDE is awful, ie. undo / redo functionality does not work as expected and typically results in permanent deletion of code
    d. Working with multiple files open and split across the screen is annoying
    e. Cannot handle files edited in other IDE's (ie. Notepad / Notepad ++ - results in corruption of line indentations and code not working properly)
    f. Poor documentation of SPIN code (ie. absolutely no documentation on *cnt* variable)
19. Planner code needs to be fixed up
    a. Multiple errors in original version of code - doesn't even compile
    b. Cryptic error messages at best
    c. Doesn't support packages
    d. Function to obtain robot poses is ambiguous and needs to be improved
    e. Code for obtaining all robot poses is unnecessarily complicated, needs improvement
    f. Integer (long) packing / unpacking needs to be fixed (high byte issue) so data wont get corrupted
    g. Needs code cleanup in general, lots of relic code
    h. Documentation would be nice
    i. Trace files are cryptic to read, could really use a better file structure that is more readable (as opposed to CSV)
20. Unable to compare floats using float math library

“If I Could Turn Back Time”

Kevin - I would have much rather taken this course after it had been fixed up, instead of having to deal with all of the hardware and software issues we have run into. It has been quite a headache. In hindsight, I would have liked to have re-written the software instead of having to deal with the issues we’ve had with the RBC communicator and the RobotTracker so that our implementation would actually work, although I should have never had to do that in the first place. Instead, we were unable to finish our final project because at the end, although our software implementation was finished and functional, literally nothing worked due to complete hardware and software failures which our implementation relied on. We wasted many hours trying to make it work. I would have also ensured that we were able to start the project sooner - a week and a half was not nearly enough time to complete the project, irrelevant of how complex our implementation was to be. Furthermore if I had known that the demo was only worth 4 marks, I would have not pulled 2 all-nighters in a row trying to get things to work, I would have instead taken my time to ensure things were done right instead of spending two days just trying to catch up on sleep and ultimately getting less work done overall. In the end, I was quite happy with how our implementation turned out despite the fact that the software and hardware it relied on didn’t function as expected.

“Robot Hardware Problems”

Kevin

1. Different cogs have a chance of arbitrarily choking out other cogs if they have no `waitcnt` delay during execution, unintuitive to figure out
2. Issues using bluetooth with cogs, very annoying
3. Robot camera colour reading does not work very well
4. Robot camera usually tracks floor, desk and walls as being blocks
5. Robot has a tendency to rock back and forth violently when using code to find blocks (not sure as to why, but not code’s fault as other students also had this problem)
6. Beeper does not run on own cog and is a blocking call, resulting in any other code being interrupted (ie. Collision avoidance not working properly due to beeper debugging)
7. Robot 3 servos actually did not work properly all
8. Bluetooth occasionally seems to get overflowed (even though it is only receiving 2 bytes per second) and will become occasionally unresponsive for up to 2 minutes, then simply resume)
9. Some robots did not properly display low batteries and would die on startup (resulting in much confusion)
10. Unsure of how `cnt` actually works, no documentation, rollovers / overflows would result in robot becoming unresponsive
11. Broken dip switches and pins for USB connection on robots were not very helpful
12. Constantly need to re-calibrate robots because servo values change
13. Tracker tags constantly come loose, no tools to fix them
14. Any incoming or outgoing robot communication data was regularly corrupted (due to the high byte issue or bluetooth noise / scrambling with cogs)
15. Block sensor would usually not go off since block was not always directly in front of the block sensor
16. DIRRS and Sonar readings were very different from each other

“Fun / Cool Stuff”