Содержание

Аннотация

В данной работе описываются понятия интерактивных задач по программированию и задач на разработку интеллектуальных агентов, функционирующих в многоагентной виртуальной среде. Основная часть работы заключается в выявлении требований, предъявляемых к системе автоматической проверки решений задач по программированию «CATS» и доработке этой системы для эффективной поддержки новых классов задач.

1. Введение

1.1. Глоссарий

Интеллектуальный агент — автономная сущность, получающая информацию о внешней среде и осуществляющая влияние на внешнюю среду, при этом их реакция рациональна в том смысле, что их действия содействуют достижению определенных параметров.

Мультиплексирование — объединение нескольких потоков данных в один.

Поток данных (англ. stream) в программировании — абстракция, используемая для чтения или записи файлов, сокетов и т. п. в единой манере. Потоки являются удобным унифицированным программным интерфейсом для чтения или записи файлов (в том числе специальных и, в частности, связанных с устройствами), сокетов и передачи данных между процессами. Поддержка потоков включена в большинство языков программирования и во все современные операционные системы. При запуске процесса ему предоставляются предопределённые стандартные потоки.

Поток выполнения (тред; от англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса.

Стандартные потоки ввода-вывода — потоки процесса, имеющие номер (дескриптор), зарезервированный для выполнения некоторых «стандартных» функций. Как правило (хотя и не обязательно), эти дескрипторы открыты уже в момент запуска задачи (исполняемого файла). В число стандартных потоков входят: стандартный ввод (STDIN), стандартный вывод (STDOUT), стандартный вывод ошибок (STDERR).

Controller (контроллер) — программа, предоставляющая виртуальную среду для интеллектуальных агентов и контролирующая их действия, запущен-

ная с помощью Spawner. Используется в многоагентных задачах.

Control режим — режим работы Spawner, если присутствует хотя бы один контроллер.

Interactor (интерактор) — программа, выполняющая роль контроллера в интерактивных задачах. От интерактора не требуется управление программой-решением через Spawner, в отличии от контроллера.

Normal (нормал) — программа, выполняющая роль интеллектуального агента и запущенная с помощью Spawner.

Pipe (пайп) — объект операционной системы, посредством которого происходит реализация передачи данных по стандартным потокам ввода-вывода.

Spawner — программа, осуществляющая контролируемое исполнение произвольных программ, используемая в CATS.

1.2. Описание предметной области

1.2.1. Соревнования по программированию

На сегодняшний день в России и мире проводится множество соревнований по программированию, включая известный командный чемпионат мира по программированию среди студентов высших учебных заведений ACM ICPC [1], а так же школьные олимпиады по информатике. Подобные соревнования получили широкое распространение и помогают относительно объективно оценить уровень знаний и умений участников в области программирования [2]. В рамках соревнований участникам предлагается набор задач, который они должны решить за определенное время, не выходя за заданные условиями задач и соревнований ограничения. Каждому участнику или команде участников предоставляется компьютер с предустановленным набором программного обеспечения, позволяющей сразу же приступить к решению задач. Некоторые соревнования допускают удалённое участие, а так же позволяют пользоваться системой автоматической проверки решений задач вне турниров — для тренировки.

При проведении соревнования по программированию, от организатора требуется разработать пакет заданий, состоящий из условий, набора тестов и эталонного решения для каждой задачи. Когда участник разработал программу и считает, что она удовлетворяет условиям задачи, он предоставляет данное решение для проверки организаторам, где, в свою очередь, проходит тестиро-

вание и сравнение результата работы программы участника с результатами работы эталонного решения. После этого делается вывод о корректности предложенного на проверку решения.

Для упрощения проведения соревнований используются системы автоматической проверки решений задач по программированию. Такие системы предоставляют интерфейсы для регистрации в турнире, получения условий, отправки условий, слежения за ходом турнира и автоматически проводят тестирование решений. Стоит заметить, что такие системы могут использоваться для автоматизации учебного процесса в дисциплинах информатики. Такая практика успешно применяется на кафедре информатики, математического и компьютерного моделирования ШЕН ДВФУ.

Последние годы в соревнования Всероссийской олимпиады школьников по информатике[11] наблюдается появление нового вида задач — основанных на игровой стратегии[22]. Задачи данного вида далее будем называть интерактивными задачами.

1.2.2. Соревнования Искусственного Интеллекта

Соревнование ИИ — мероприятие, посвящённое одновременному взаимодействию пользовательских программ, выполняющих роль интеллектуальных агентов в некоторой многоагентной среде. Среда может быть как реальной — например соревнования роботов, так и виртуальной, предоставляемой некоторым программным обеспечением. В данной работе нас интересует только последний случай. Данное взаимодействие осуществляется с целью выявления победителя — агента, который лучше остальных ведёт себя в данной среде, согласно условиям задачи, которая может не иметь объективно лучшего решения или же данное решение не может быть найдено за приемлемое время.

Сложность таких симуляций, обусловленная протяжённостью во времени в смысле пошагового выполнения и одновременного взаимодействия более одного агента, порождает необходимость использования программ визуализаторов хода турнира, для наглядности происходящего. На скриншотах ниже представлены примеры визуализаторов некоторых существующих популярных турниров ИИ.

Задачи, ставящиеся на таких соревнованиях будем называть интерактивными многоагентными, или просто многоагентными.

На рис. 1 представлен визуализатор соревнования Robocode — в ходе которого на поле соревнуются танки. Задача каждого танка - выжить и уничтожить как можно больше противников. Каждый танк — управляется программой-агентом. На рис.2 представлен визуализатор некоторого стратегического соревнования, заключающегося в пошаговом захвате территории. Некоторые системы предоставляют более одного типа соревнований и возможность разрабатывать программу-агент в веб-браузере Рис. 3.

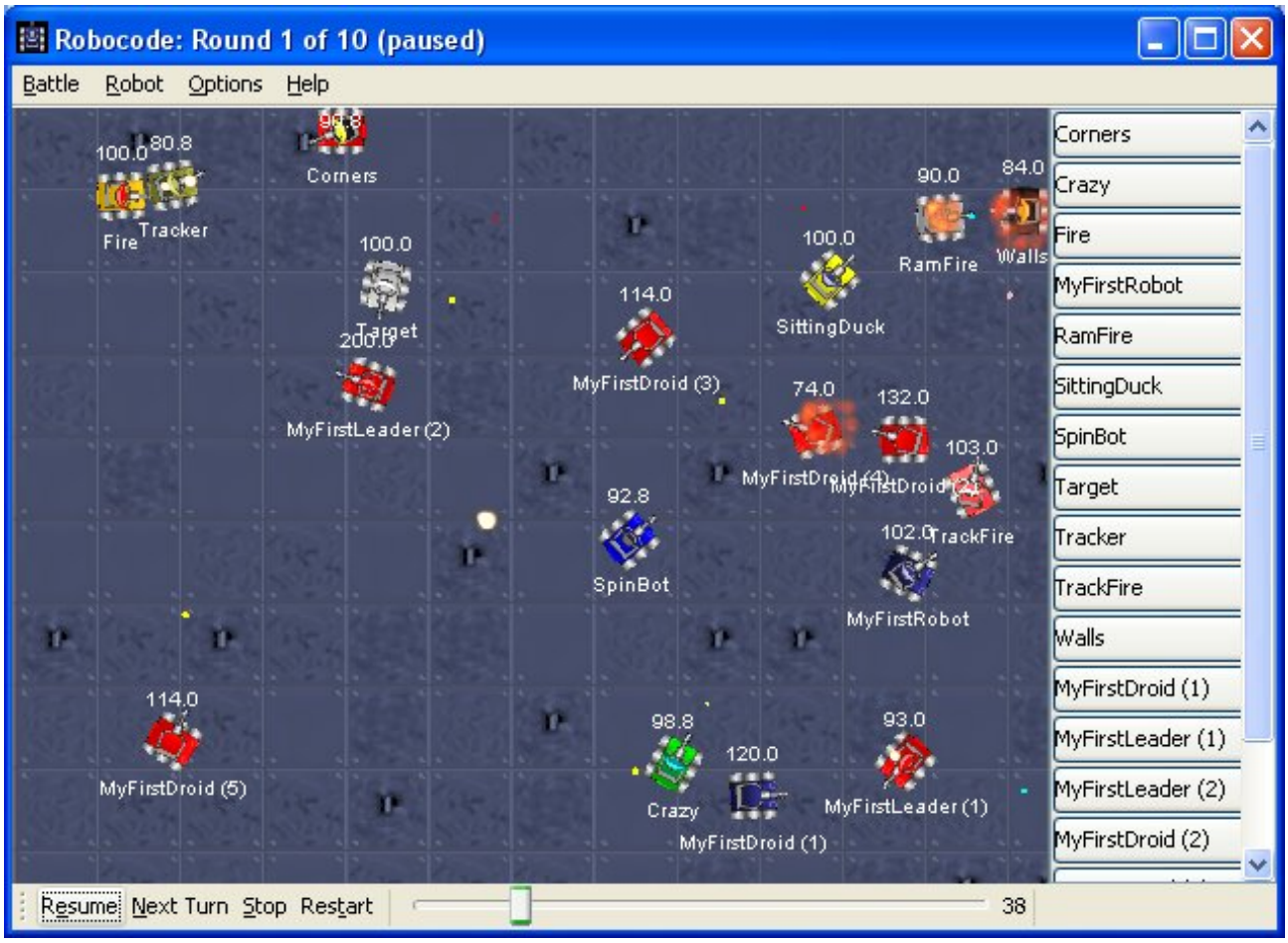

Рис. 1: Robocode

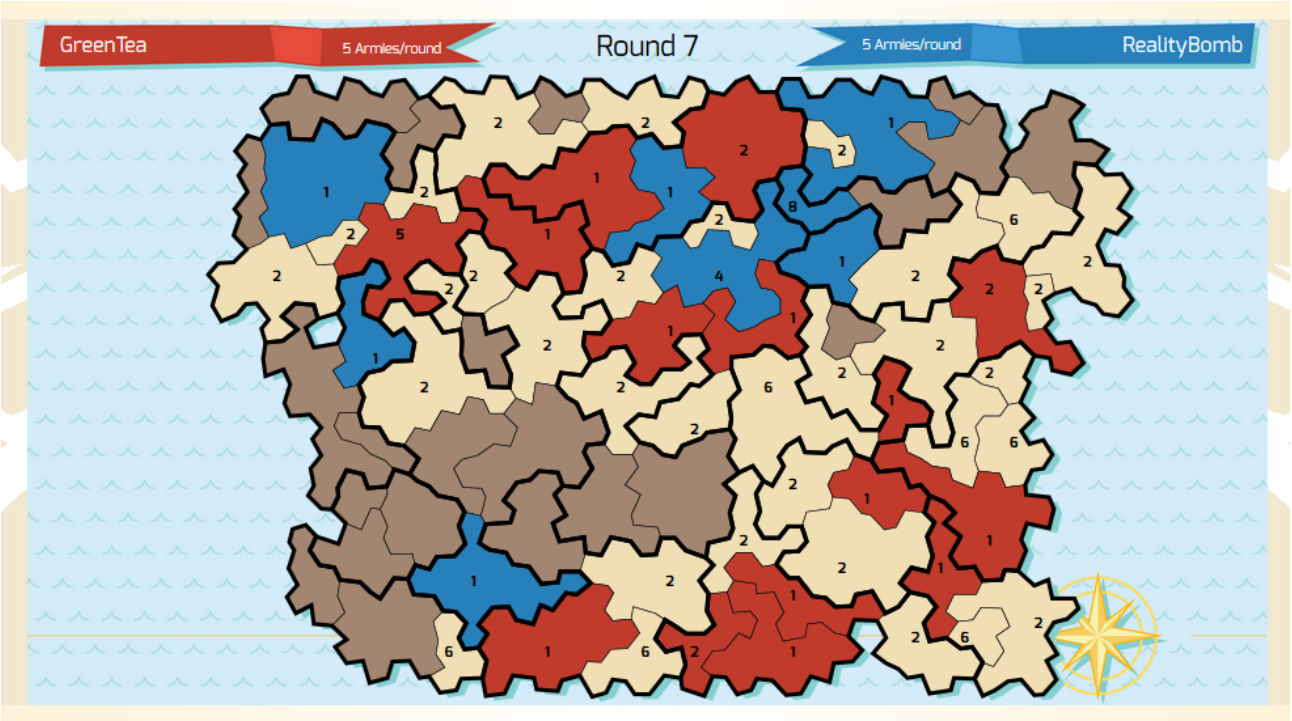

Рис. 2: Warlight

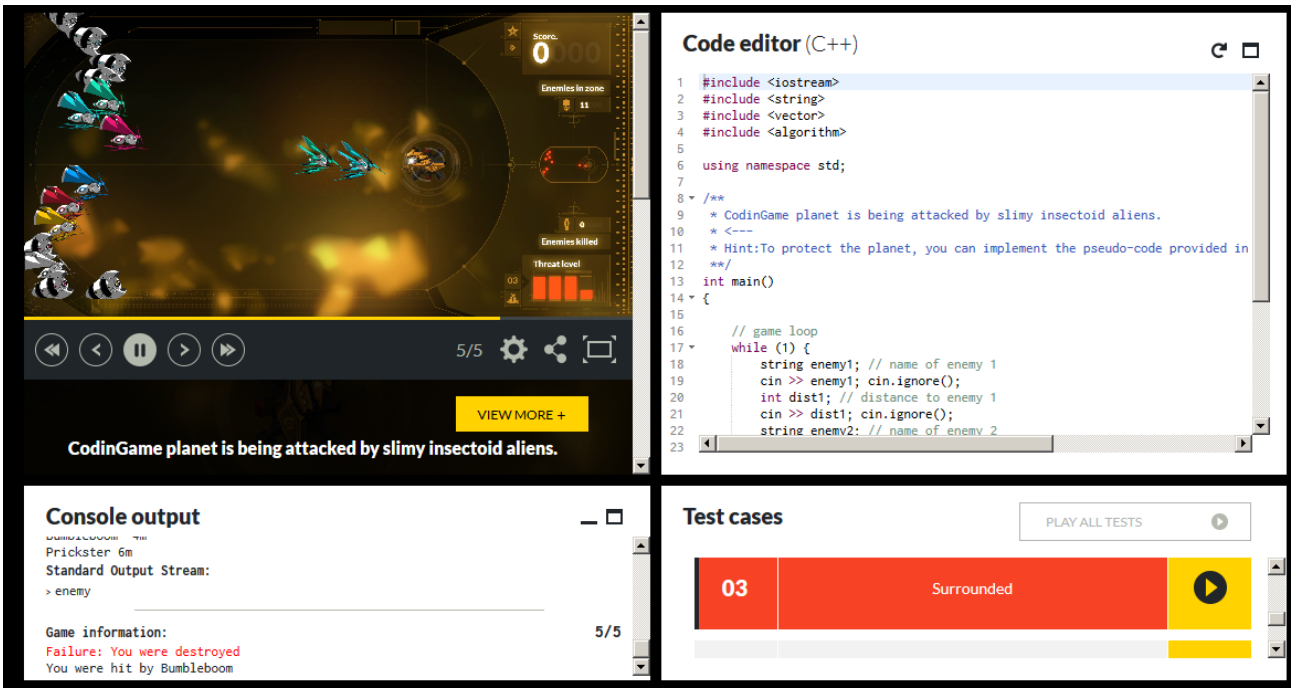

Рис. 3: codingame.com

1.2.3. Система CATS

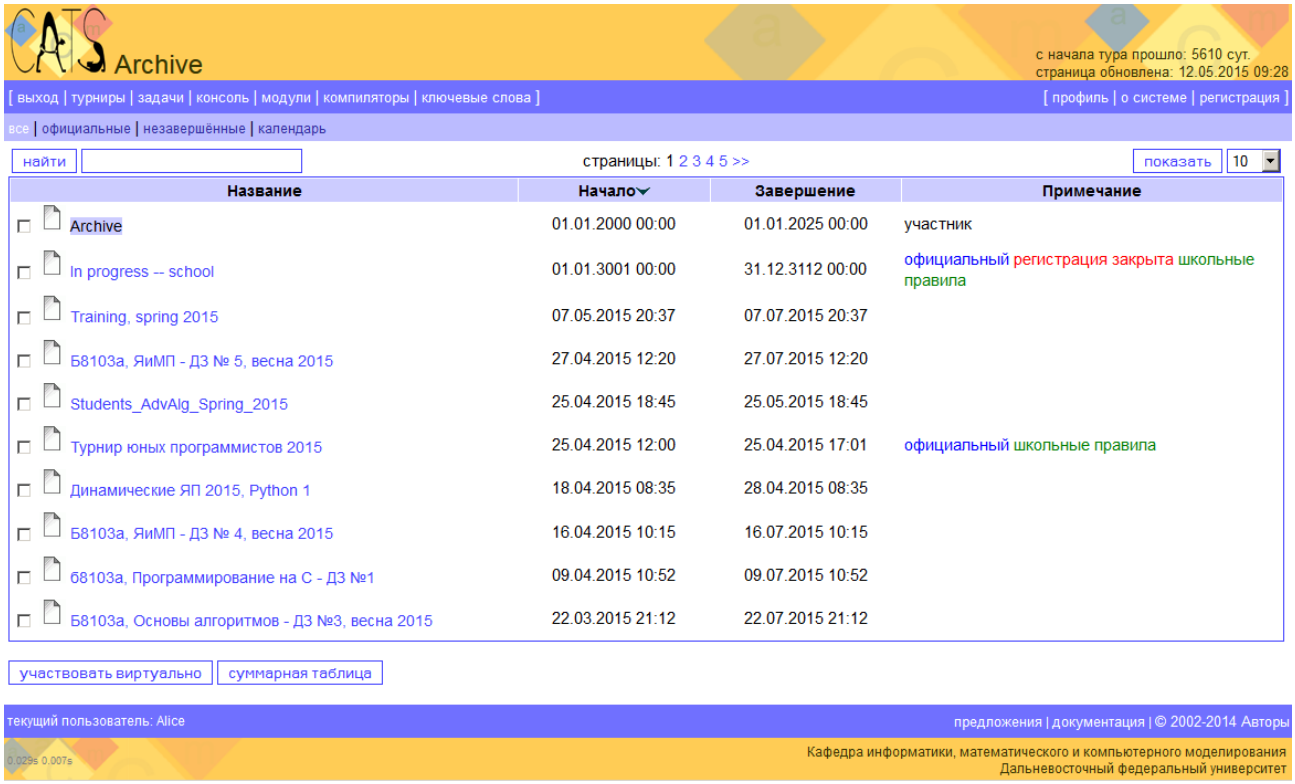

Рис. 4: Веб-интерфейс системы

Одной из систем автоматизации организации соревнований по программированию является CATS [3, 4] (Рис. 4). Система разрабатывалась на базе ДВФУ и успешно применяется при проведении соревнований начиная от школьных олимпиад и заканчивая четвертьфиналом чемпионата мира по программированию ACM ICPC. История системы начинается с 2002 года и продолжается до сих пор. За это время на базе CATS было проведено огромное количество соревнований, система неоднократно улучшалась и дополнялась [4, 5, 6, 7, 8, 9].

CATS включает в себя такие компоненты как веб-интерфейс — cats-main, базу данных, компонент тестирования решений — cats-judge [4], а так же компонент контролируемого исполнения программ Spawner (подробнее см. раздел Архитектура).

Веб-интерфейс предоставляет возможность участвовать в доступных турнирах, просматривать задачи, отправлять соответствующие решения, выполненные в разрешенных системой (турниром) средах разработки, следить за ходом их тестирования, следить за ходом турнира и др. Для организаторов, в

свою очередь, веб-интерфейс позволяет создавать турниры, добавлять в них задачи, следить за ходом турнира и др.

База данных обеспечивает связь между веб интерфейсом и компонентом cats-judge, а так же хранит информацию о существующих в системе турнирах, участниках, задачах, решениях и пр.

Веб интерфейс и база данных являются кросс-платформенными составляющими системы и могут быть установлены на компьютеры под управлением различных операционных систем.

Компонент тестирования решений отвечает за контролируемый запуск и тестирование программ решений участников. Во время исполнения программ от judge требуется чтобы они не нарушали набора ограничений на системные ресурсы, наборы правил безопасности, а в случае некорректного хода работы — информация о произошедших ошибках должна быть корректно обработана и учтена. После чего результаты о ходе тестирования и запуска решения участника отправляются в базу данных.

Компонент контролируемого исполнения используется при тестирования решений. Предоставляет обеспечение безопасности, управление вводом/выводом и контроль за потребляемыми программой ресурсами, такими как память, время. Для задачи контролируемого исполнения в cats-judge используется компонент Spawner[10], представляющий собой отдельный программный продукт абстрагированный от предметной области спортивного программирования. Большая часть изменений вносимых в систему CATS в рамках данной работы касается непосредственно компонента Spawner.

Компонент Spawner имеет необходимый для поддержки интерактивных задач функционал, но поддержка данного типа задач не реализована в других компонентах CATS. В случае многоагентных задач необходима доработка компонента Spawner.

1.2.4. Задачи

Рассмотрим разницу между стандартными, интерактивными и многоагентными задачами в смысле потоков обмена данными и количества участников об-

мена данными в контексте CATS. На иллюстрациях ниже Н.П. — нормальная программа (решение стандартной или интерактивной задачи, либо интеллектуальный агент), У.П. — управляющая программа (интерактор или контроллер).

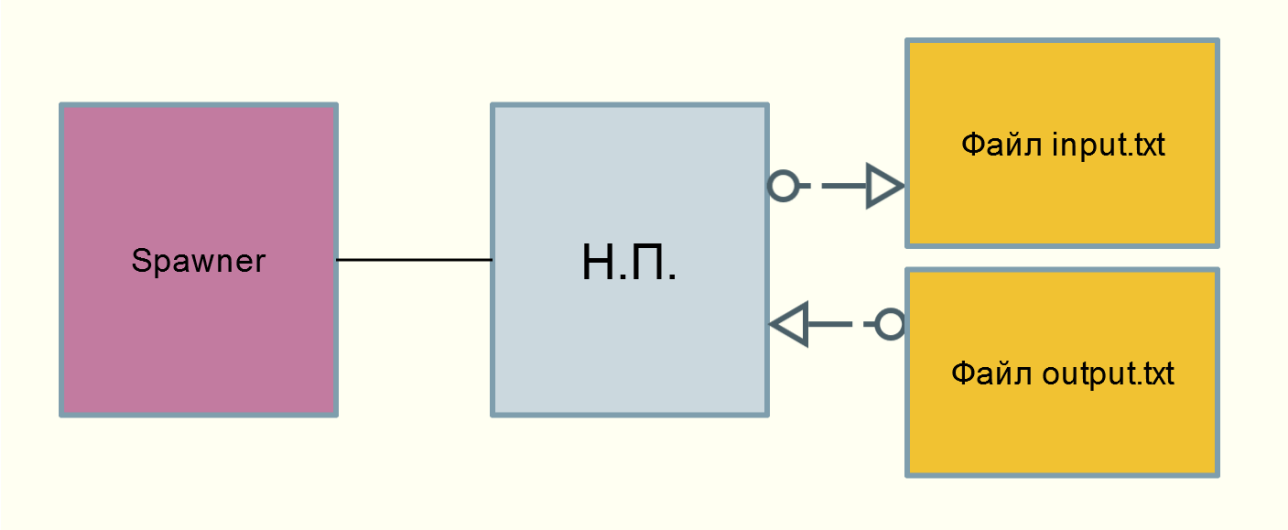

Рис. 5: Потоки данных в стандартной задаче

Последовательность действия для проверки стандартной задачи включает в себя взаимодействие тестирующей системы и решения задачи (рис. 5). Входные и выходные данные чаще всего передаются через файлы, указанные в условиях задачи или через стандартные потоки STDIN, STDOUT.

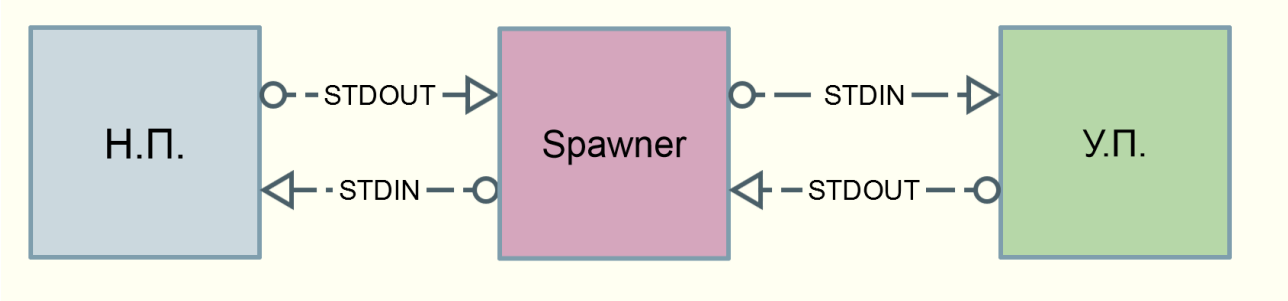

Рис. 6: Потоки данных в интерактивной задаче

При использовании игровой стратегии, проверка решения рассматривается как игра двух противников — тестирующей системы (интерактор) и тестируемой программы (решение участника). Первый ход осуществляет тестирующая система (проверяющая программа). Далее ход переходит к тестируемой программе. Она запускается на входных данных, подготовленных тестирующей системой, и выдает некоторый результат. Ход вновь получает тестирующая система. Она анализирует результат, полученный тестируемой программой. На

основе результатов анализа принимается решение о продолжении тестирования. Если тестирование продолжается, то проверяющая система подготавливает новый набор входных данных, при этом может быть использована информация , полученная от тестируемой программы, на предыдущих шагах. После чего ход снова передается тестируемой программе, и так далее (см. Рис. 6). [22]

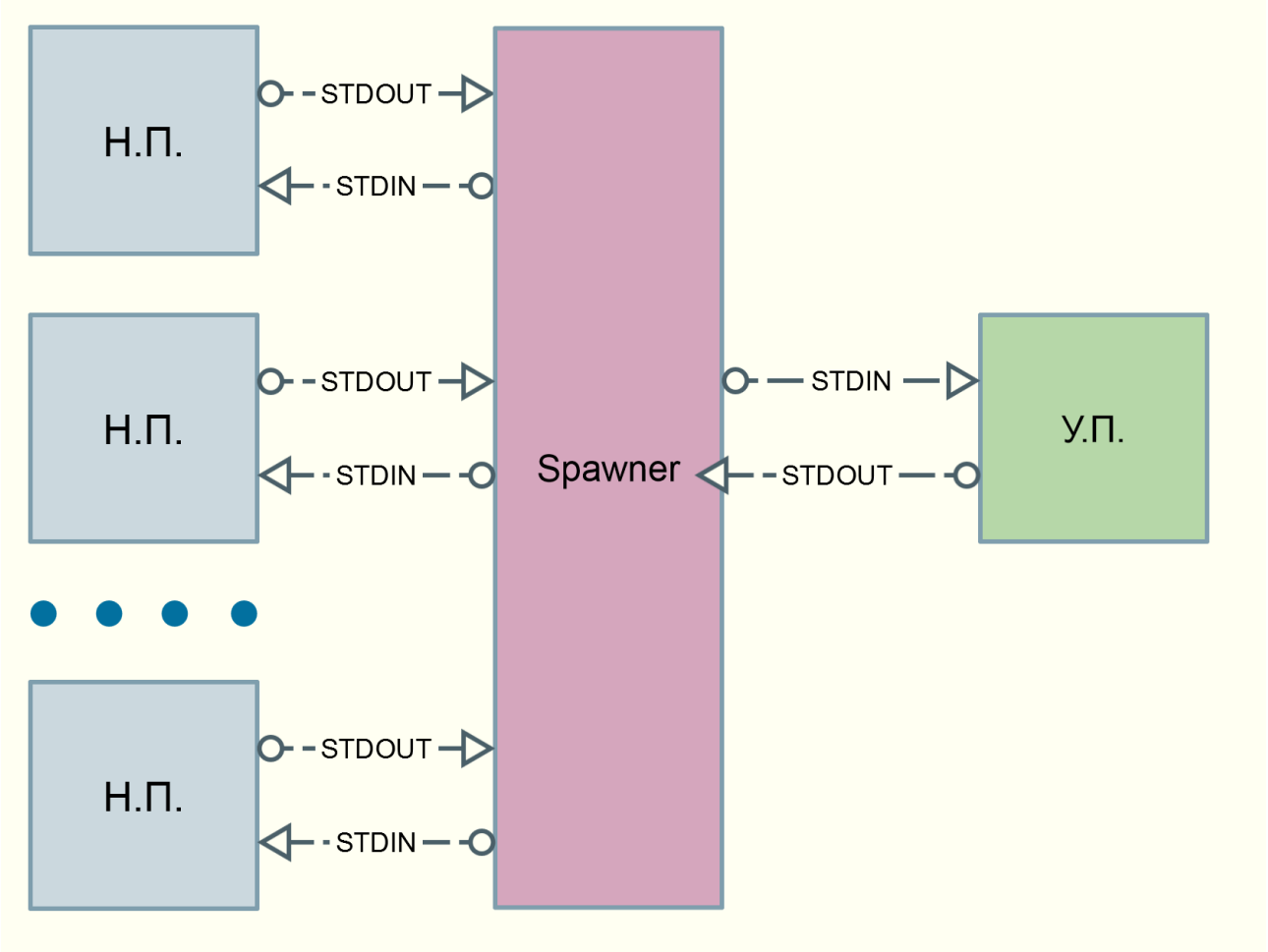

Рис. 7: Потоки обмена данных в многоагентных задачах

В случае многоагентных задач увеличивается количество одновременно взаимодействующих программ и у тестирующей программы (контроллера) появляется необходимость управлять исполнением нормальных программ через Spawner. При этом взаимодействие с контроллером по прежнему осуществляется через два стандартных потока ввода-вывода, что наталкивает на мысль о необходимости мультиплексирования каналов связи по некоторому протоколу. (рис. 7).

1.3. Неформальная постановка задачи

Целью данной работы ставится расширение поддерживаемых CATS классов задач:

- Использование интерактивных задач в CATS
- Расширение Spawner для поддержки многоагентных задач

1.4. Обзор существующих методов решения

1.4.1. Системы организации соревнований

В рамках данной работы было проведено изучение подобных систем организации соревнований по программированию.

ejudge — это система для проведения различных мероприятий, в которых необходима автоматическая проверка программ. Система может применяться и применяется для проведения олимпиад, поддержки учебных курсов и т.д. Система предоставляет веб-интерфейс администратора и участника турнира, а так же доступ к серверам турниров из командной строки. [12]

PCMS2 (Programming Contest Management System) — Используется во время проведения Всероссийского этапа ACM ICPC. [13]

Contester — это система для проведения турниров и индивидуального решения задач по олимпиадному программированию (спортивному программированию). Система содержит условия задач - от легких до олимпиадных - и возможность проверки решений на большинстве современных языков: C++, Object Pascal, Java и языках .NET: C#, J# и Visual Basic.

Contester работает на Windows и на Linux. Язык реализации Delphi/freepascal. Представляет собой полноценную систему организации соревнований готовую к запуску с момента установки и распространяется в виде установочного файла или архива, в зависимости от операционной системы. [14]

PC^2 (Programming Contest Control System) — система разработанная в Калифорнийском Государственном Университете Sacramento (CSUS) в поддержку соревнований по программированию, проводимых ACM и, в частности, ACM International Collegiate Programming Contest (ICPC) и его региональных этапов. [15]

DOMjudge — автоматизированная тестирующая система, для проведения соревнований по программированию, подобных ACM ICPC. Основной упор сделан на удобство использования и безопасность. Система применялась во многих соревнованиях, распространяется свободно на условиях открытого ПО. Для контроля исполнения используется набор небольших консольных утилит. [17]

dudge — это универсальная система для проведения олимпиад по программированию и другим предметам, написанная на Java и J2EE с использованием СУБД PostgresSQL и распространяющаяся по лицензии GPL. [18]

Название	Интерактивные задачи	Соревнования ИИ	Лицензия	Последняя версия
ejudge	Да	Нет	GPL	3.3.0, 15.01.2014
PCMS2	Да	Нет	Проприетарная	2, 17.11.2004
Contester	Нет	Нет	Проприетарная	2.4, 07.11.2010
PC^2	Да	Нет	Проприетарная	9.2.4, 20.09.2014
DOMjudge	Да	Нет	GPL	4.0.4, 27.10.2014
dudge	Нет	Нет	GPL	GIT-master

Таблица 1: Сравнение доступных систем контроля исполнения программ

1.4.2. Системы искусственного интеллекта

Скриншоты некоторых таких системы были приведены в 1.2.2.. Было рассмотрено несколько таких систем. В результате было выявлено, что подавляющее их большинство (например Robocode[25], Rock Paper Scissors[28], Vindinium[27] Code Cup[30], Google AI Contests) специализированно для одной конкретной задачи. Те из них, что поддерживают множество задач (Hacker Rank[31], Coding Game[29]) не предоставляют пользователям возможность разработки задач и являются закрытыми, что затруднило подробный анализ.

1.4.3. Системы контролируемого исполнения

Средства контролируемого исполнения применяются во многих областях, например Online IDE, Тестирующие системы, антивирусы, Continuous Integration. В рамках данной работы нас интересует поддержка многозадачности в таких системах, что означает не только возможность одновременно запускать несколько процессов но и предоставить возможность некоторым процессам взаимодействовать с системой контролируемого исполнения и другими процессами, а так же предоставляют возможность одним контролируемым процессам управлять выполнением других.

Существующие средства не предоставляют широких возможностей настройки взаимодействия процессов через стандартные потоки, а так же не предоставляют интерфейса контроля других процессов.

Среди прочих стоит выделить программный комплекс Docker, который позволяет настраивать отображение портов контейнеров, в которых запускаются процессы, связывая их и тем самым позволяя общаться друг с другом. [32]. Данный подход обеспечивает только обмен данными между процессами, при чём посредством сетевого взаимодействия, что нас не устраивает.

1.4.4. Вывод

Большинство систем автоматизации проведения соревнований и тестирования решений задач имеют поддержку интерактивных задач, но не было обнаружено ни одной, имеющей встроенную возможность проведения соревнований ИИ. Добавление в CATS поддержки интерактивных задач позволит рас-

ширить класс поддерживаемых задач и использовать в CATS интерактивные задачи с проводимых в других системах турниров.

Существующие системы соревнований ИИ узконаправленны или предоставляются на платной основе как сервис, без возможности самостоятельной разработки новых задач.

Требуется значительная доработка модуля Spawner для того чтобы интеграция турниров ИИ в CATS стала возможной.

2. Требования к окружению

2.1. Требования к программному и аппаратному обеспечению

Для сборки проекта Spawner требуется система cmake версии не менее 2.8. Компиляция Spawner требует наличие компилятора языка C++11, поддерживаемого системой cmake. Был протестирован компилятор входящий в состав Visual Studio 2013, а так же GCC 4.9. Spawner функционирует только на системах семейства Windows, начиная с Windows XP. Полная поддержка доступна начиная с Windows Vista, что связанно с использованием функции WinAPI CancelSynchronousIo, необходимой для отмены синхронных операций ввода-вывода в пределах потока.

Требования к остальным компонентам системы рассмотрены в [4].

3. Архитектура системы

CATS является аппаратно-программным комплексом, состоящим из нескольких компонент: веб-интерфейс, база данных, тестирующий компонент cats-judge и модуль контроллируемого исполнения Spawner.

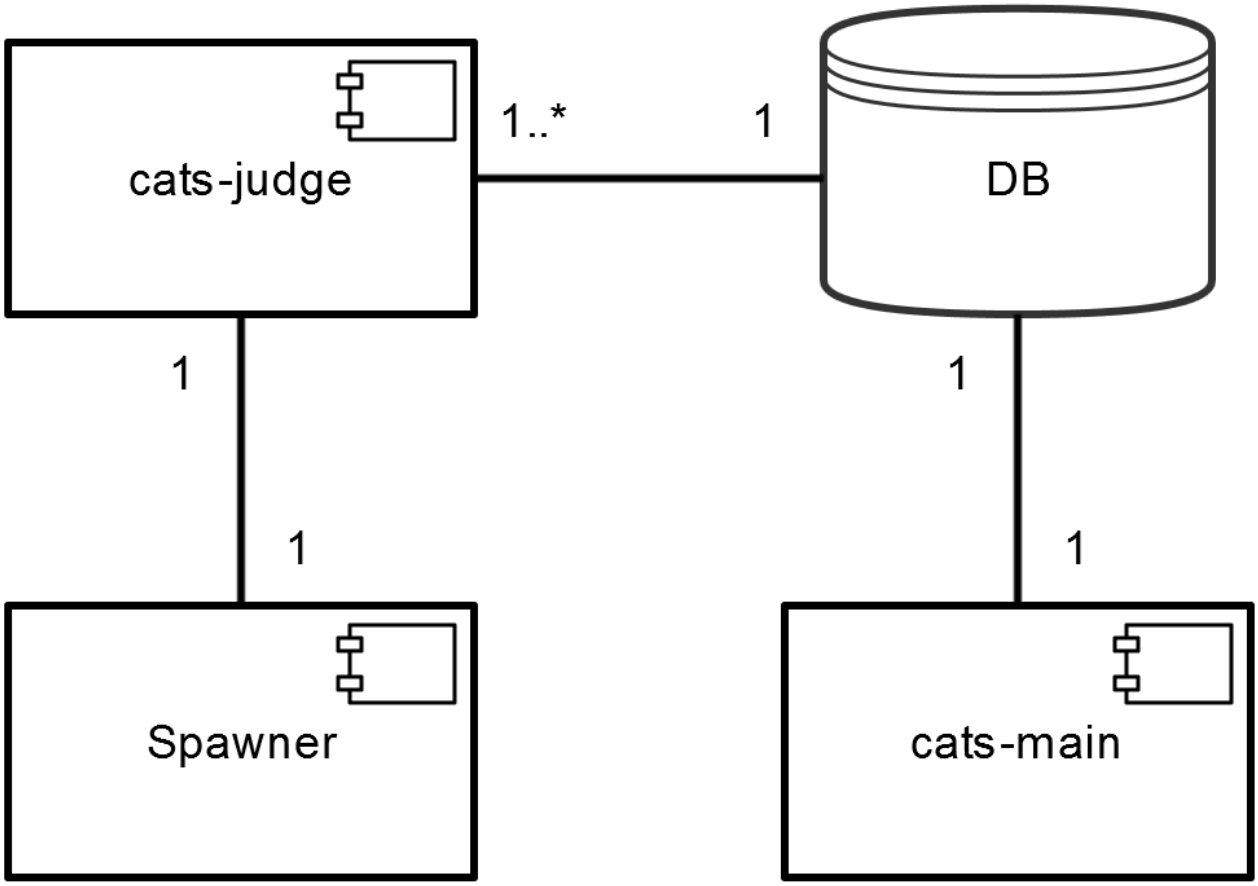

Рис. 8: Компоненты CATS

3.0.1. Веб-интерфейс

Веб-интерфейс [4] является кросс-платформенной частью системы. Веб сервер может быть запущен практически на любой системе. Фактически, с помощью веб-интерфейса осуществляется управление содержимым базы данных.

3.0.2. База данных

В базе данных [4] системы хранятся данные о проводимых турнирах, пакеты задач, исходный код, информация об участниках, их попытках, а так же информация по средам разработки. Однако привязка модуля judge к какому-либо виду базы данных усложняет его развертывание на конечной машине.

3.0.3. Компонент cats-judge

cats-judge[4] содержит загруженный кэш задач, кэш попыток, различные конфигурационные файлы и модуль Spawner.

Обычно, для сокращения нагрузки одновременно запущено несколько компьютеров с judge. Они подключаются к базе данных и распределяют задачи на исполнение между собой.

В случае отсутствия задачи в кэше происходит загрузка из базы данных, генерация тестов и т.д.

В общем случае коммуникация между модулями проходит в следующем порядке:

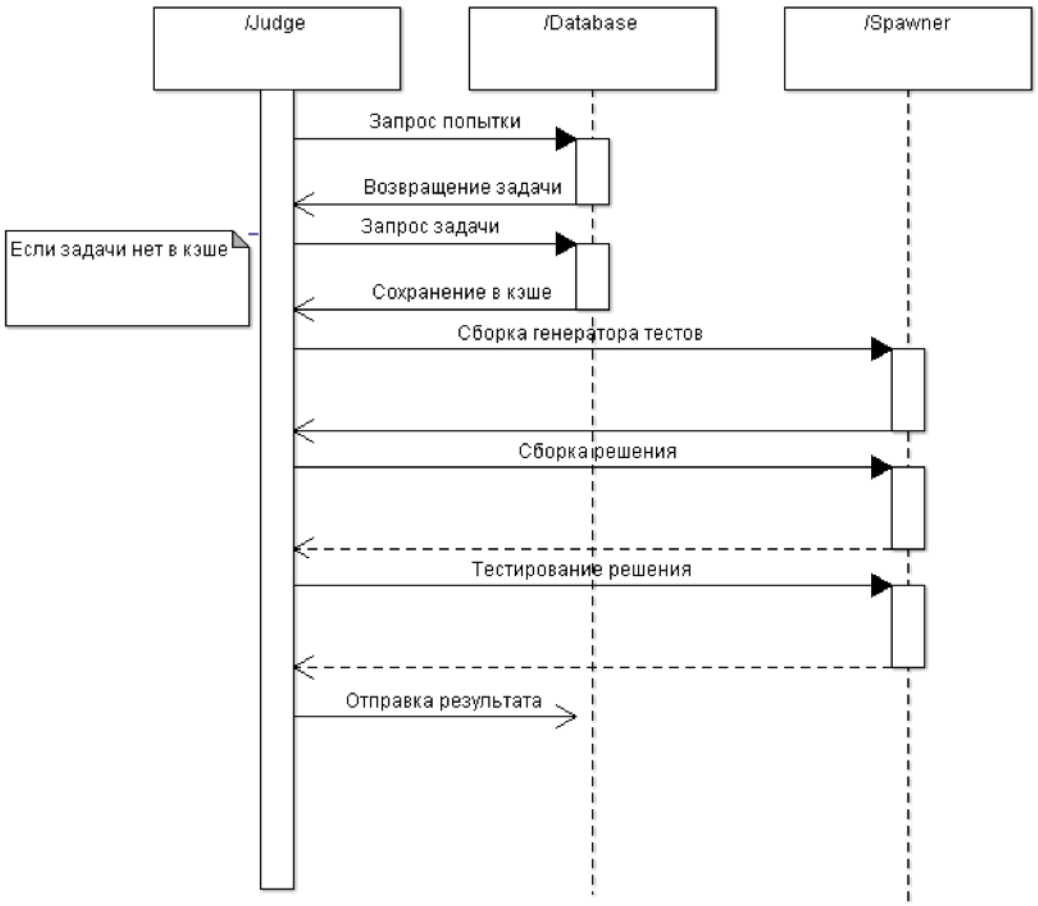

Рис. 9: Схема работы cats-judge

3.0.4. Модуль Spawner

Во время проведения тестирования требуется создать для программы изолированную среду и задать необходимые ограничения на ресурсы. Так же в случае некорректного хода исполнения тестируемой программы требуется опреде-

лить характер и возможную причину ошибок и должным образом отреагировать на это. Эти задачи решает компонент Spawner.

В данной работе важно, что Spawner так же позволяет связывать друг с другом стандартные потоки запускаемых процессов. На рис. 10 изображена схема связи стандартных потоков ввода-вывода для случая двух процессов, с акцентом на внутреннее устройства. Такого типа связи могут быть установленны между любыми двумя стандартными потоками ввода и вывода любых процессов, одновременно находящихся в контролируемом исполнении. Таким образом схема может быть расширена на произвольное количество процессов. Каждому входному и каждому выходному потоку соответствует поток исполнения Spawner. Поток чтения следит за появлением данных в STDOUT пайпе и перенаправляет их в своё множество буферов. Это множество буферов может пересекаться с множеством буферов потока записи, так как каждый буфер связывает ровно два потока. Поток записи следит за появлением данных в своих буферах и перенаправляет их в STDIN пайп.

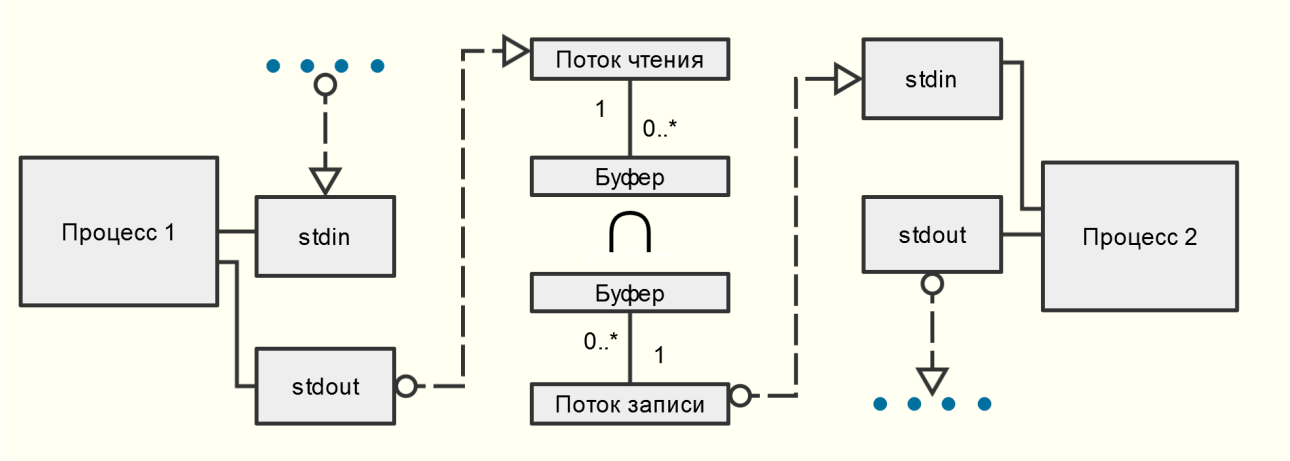

Рис. 10: Связь стандартных потоков

4. Спецификация данных

4.1. Аргументы командной строки Spawner

Аргументы командной строки Spawner подробно описаны в [9]. В рамках данной работы стоит рассмотреть способ запуска нескольких процессов, а так же формат связи потоков ввода-вывода запускаемых процессов.

– необходимо указать последовательность символов — разделитель, которая будет отделять секции аргументов командной строки для каждого запускаемого процесса. Это осуществляется с помощью ключа –separator:

```
sp.exe --separator=//
--// p1.exe
--// p2.exe
--// p3.exe
--// p4.exe
```

Таким образом будет запущено 4 процесса. Для настройки параметров процесса нужные флаги должны помещаться в соответствующую секцию.

– для того, чтобы осуществить связь потоков ввода вывода необходимо использовать флаги –in=<name> для STDIN и –out=<name> для STDOUT, где <name> имеет формат *<число>.stdin или *<число>.stdout, что позволяет сослаться на соответствующие потоки запускаемого процесса с индексом <число>. Процессы нумеруются в порядке их передачи Spawner, начиная с 0. Так же может быть указано значение –in=std или –out=std, что осуществляет перенаправление на стандартный ввод/вывод. Следующая командная строка связывает STDOUT контроллера с STDIN каждого нормала и STDIN контроллера с STDOUT каждого нормала:

```
sp.exe --separator=//
--// controller.exe
--// --in=*0.stdout --out=*0.stdin normal-1.exe
--// --in=*0.stdout --out=*0.stdin normal-2.exe
--// --in=*0.stdout --out=*0.stdin normal-3.exe
```

– для того чтобы пометить запускаемый процесс как контроллер требуется передать ключ –controller:

```
sp.exe --controller controller.exe
```

4.2. Протокол обмена данными

Рассмотрим отличие обычных задач, от интерактивных и многоагентных.

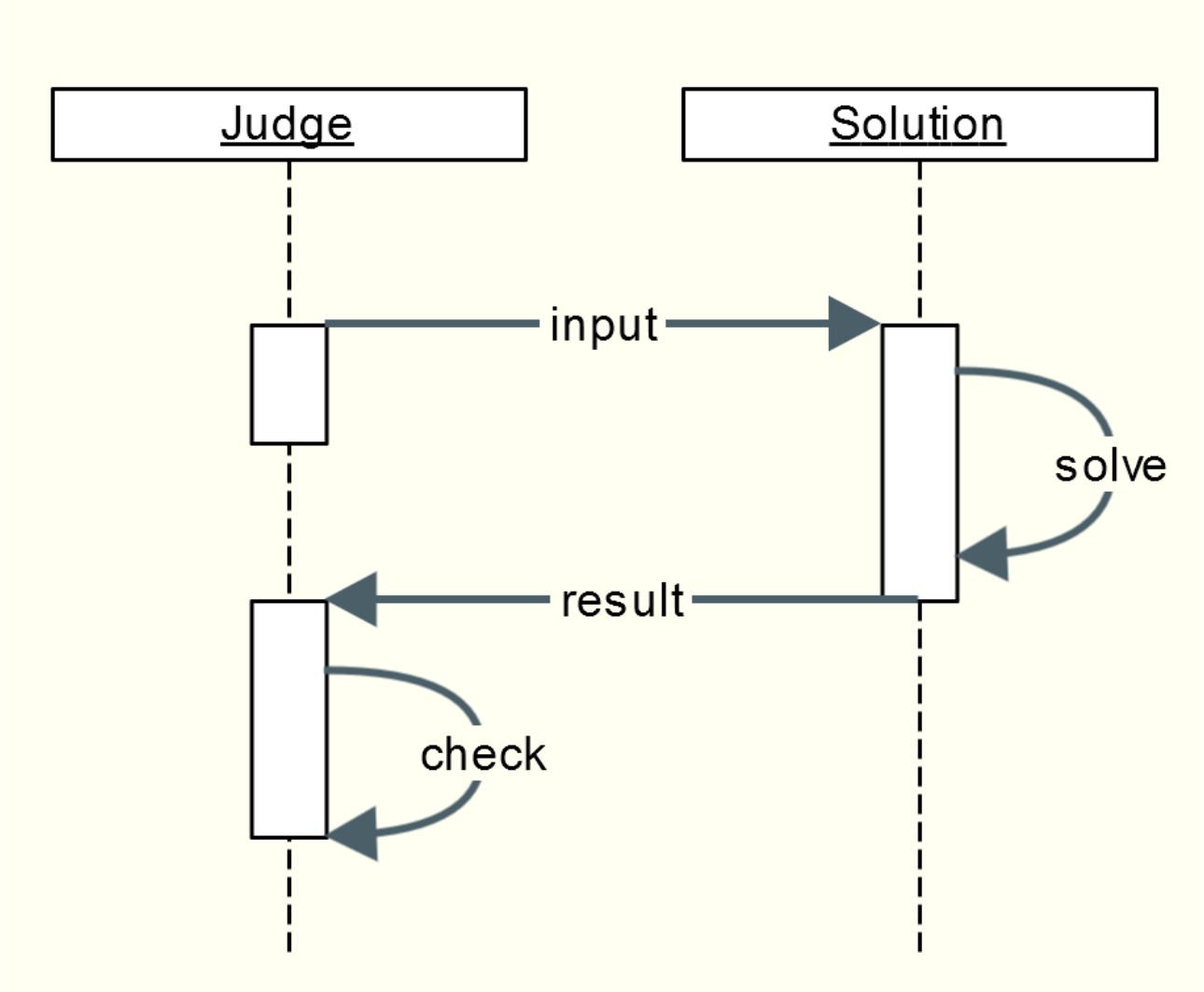

Рис. 11: Последовательность проверки обычной задачи

Последовательность действия для проверки обычной задачи включает в себя взаимодействие judge и решения задачи (рис. 11).

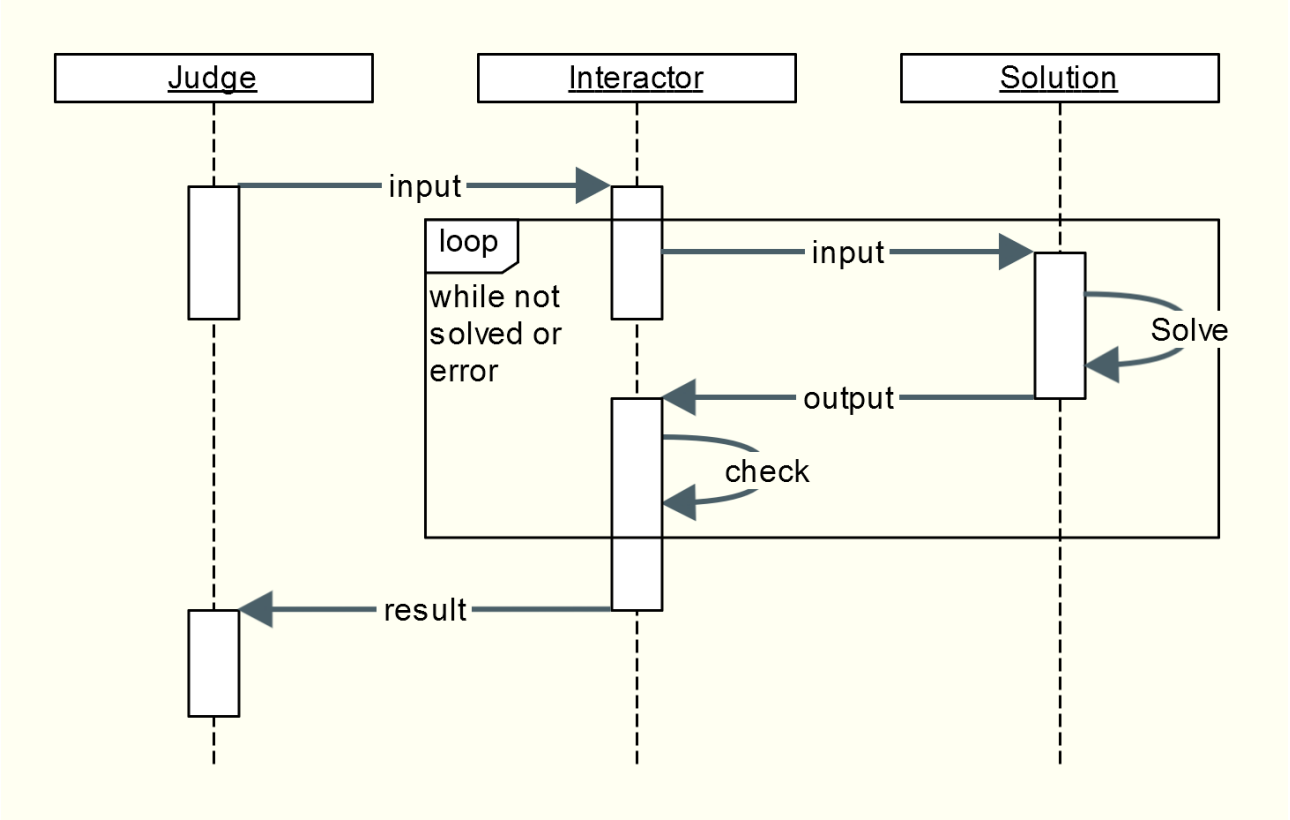

Рис. 12: Последовательность проверки интерактивной задачи

В случае же интерактивных задач появляется необходимость в дополнительной программе - интеракторе. Происходит обмен сообщениями между интерактором и решением в цикле (рис. 12).

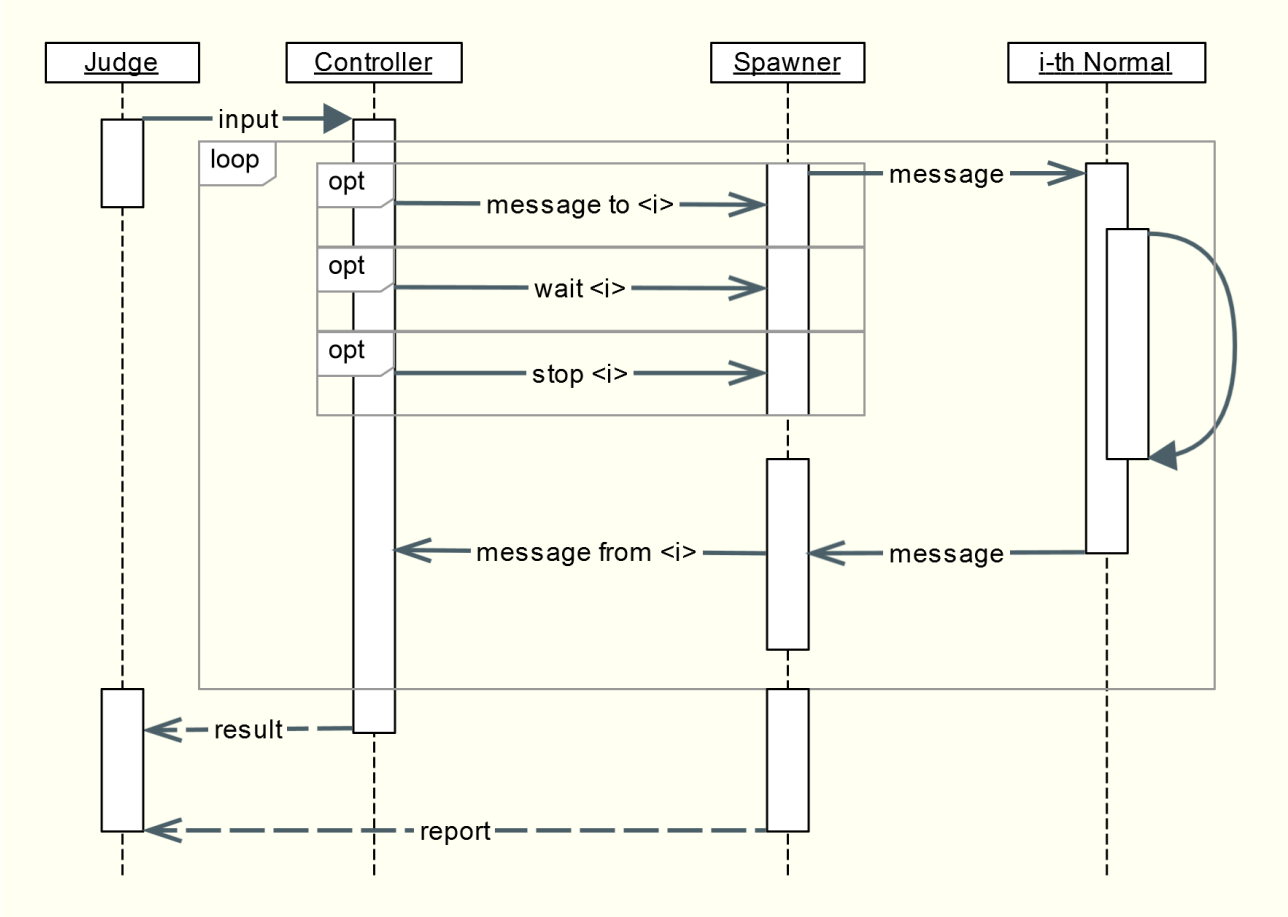

Рис. 13: Последовательность действия в режиме турнира ИИ

Для многоагентных задач взаимодействие со Spawner перестаёт быть прозрачным и учитывается тот факт, что одновременно происходит взаимодействие интерактора и нескольких решений задачи (рис. 13).

Данный протокол был разработан для мультиплексирования выходных потоков нескольких нормалов в один канал связи — стандартный входной поток контроллера, а так же демультиплексирования выходного потока контроллера во входные потоки нескольких нормалов, что позволяет контроллеру определить от какого нормала пришло сообщение, а так же доставить сообщение от контроллера указанному нормалу.

Целью данного протокола так же является возможность контроллера управлять выполнением нормалов — он может указать ожидание ввода от нормала, что выведет процесс нормала из suspended режима, а так же остановить процесс любого нормала, посредством отсылки Spawner специальных сообщений.

В контексте предметной области данный протокол будет использован для реализации поддержки многоагентных задач. Для интерактивных задач верным остаётся только необходимость сброса стандартного потока вывода, как для ин-

тератора, так и для нормала, а так же тот факт, что разделителем сообщений является символ перевода строки.

4.2.1. Схема запуска

Контроллер и нормал запускаются с помощью Spawner, передаваемые в качестве аргументов командной строки. Предъявляются следующие требования:

- контроллер должен быть помечен флагом –controller.
- контроллер должен быть единственным
- наличие контроллера должно переводить spawner в режим управления
- остальные требования применимы только в режиме управления
- Количество нормалов должно быть передано контроллеру первым аргументом командной строки
- stdin каждого нормала должен быть соединён с stdout контроллера с помощью –in=*0.stdout
- stdout каждого нормала должен быть соединён с stdin контроллера с помощью –out=*0.stdin

Пример командной строки запуска:

```
1 sp.exe --json -sr=report.json -hr=1 --separator=//
2   --// --controller --out=std controller.exe 3
3   --// --out=std --in=*0.stdout --out=*0.stdin normal-1.exe
4   --// --out=std --in=*0.stdout --out=*0.stdin normal-2.exe
5   --// --out=std --in=*0.stdout --out=*0.stdin normal-3.exe
```

4.2.2. Правила Взаимодействия

- все нормалы должны запускаться в приостановленном (suspended) состоянии
- Spawner должен продолжать (resume) выполнение указанного нормала, в ответ на запрос ожидания данного нормала от контроллера

– Spawner должен приостанавливать выполнение нормалов после получения от них сообщения, если контроллер ожидал ответа от данного нормала

– контроллер может послать сообщение конкретному нормалу

– контроллер может послать Spawner запрос на ожидание сообщения от конкретного нормала

– нормал может послать сообщение контроллеру

– контроллер может послать Spawner запрос на полную остановку указанного нормала

– ограничения времени контроллера должны быть сброшены после каждого сообщения, посланного контроллером

– ограничения времени нормала должны быть установленны в ответ на запрос ожидания данного нормала контроллером

– ограничения времени нормала должны быть сброшены при отсылке нормалом сообщения

4.2.3. Сообщения

Сообщения имеют форму <заголовок><тело><перевод строки> Тело может состоять из любых символов кроме символов перевода строки или возврата каретки. Заголовок должен всегда кончаться символом #. Перед символом # может быть латинская буква, перед которой в свою очередь может быть целое число.

– контроллер и нормал должны отсылать сообщения через stdout

– stdout должен быть сброшен (flushed) после отсылки сообщения

– сообщения должны заканчиваться символом перевода строки

4.2.4. Примеры сообщений

Два сообщения:

```
printf("First message\nSecond message\n");
fflush(stdout);
```

Одно сообщение:

```
printf("\n");
fflush(stdout);
```

Забыт перевод строки:

```
printf("foo");
fflush(stdout);
```

Не сброшен буфер:

```
printf("foo\n");
```

4.2.5. Сообщения контроллера

Контроллер может отсылать два типа сообщений — сообщения для нормала и сообщения для Spawner.

При отсылке сообщений нормалу, контроллер обязан указать в заголовке сообщения индекс нормала, которому предназначено сообщение. Нормалы индексируются в порядке их передачи Spawner в качестве аргументов командной строки, начиная с единицы.

Например для такой схемы запуска:

```
 sp.exe --separator=//
 --// --controller controller.exe 3
 --// --in=*0.stdout --out=*0.stdin normal-1.exe
 --// --in=*0.stdout --out=*0.stdin normal-2.exe
 --// --in=*0.stdout --out=*0.stdin normal-3.exe
```

Чтобы отослать сообщение ”Foo”приложению normal-1.exe контроллер должен выполнить следующий код:

```
printf("1#Foo\n");
fflush(stdout);
```

Если контроллер указывает неверный индекс, то Spawner обязан ответить контроллеру сообщением об ошибке:

```
<i>I#\n
```

где <i> — это индекс неверного normal.

Контроллер может сообщить Spawner об ожидании им ответа от конкретного нормала: <i>W#<перевод строки>

Контроллер может запросить Spawner прекратить выполнение процесса указанного нормала с помощью сообщения <i>S#. Spawner не должен сообщать об ошибке на запрос остановки, если индекс неверен. Например чтобы запросить остановку нормала с индексом 4:

```
printf('4S#\n');
fflush(stdout);
```

Сообщения отсылаемые по индексу 0 считаются сообщениями, отсылаемыми spawner. Дополнительный латинский символ может уточнить значение сообщения. Этот случай зарезервирован для последующих версий протокола.

4.2.6. Сообщения нормала

Spawner обязан дополнить слева сообщения normal его индексом и символом # чтобы образовать заголовок сообщения, который будет разобран контроллером.

Например:

```
// let this normal's index be 3
printf("12 45 56 65\n");
fflush(stdout);
// controller will get "3#12 45 56 65\n"
```

4.3. Формат пакета задачи

Описание конфигурации задачи представляет собой XML файл. Формат этого файла в полной мере описан в документации к CATS. [33]

В данной работе в рамках добавления поддержки интерактивных задач в формат заданий добавляется тэг Run для тэга верхнего уровня Problem. Данный тэг имеет атрибут method, принимающий значения interactive или default.

```
<Run method="interactive" />
```

4.4. Конфигурация модуля cats-judge

Настойка доступных сред разработки и способов запуска для этих сред осуществляется с помощью XML файла config.xml находящегося в корневом каталоге cats-judge. Правила заполнения этого файла см. в [4]. В данной работе в config.xml были добавлены следующие примитивы:

– run_interactive - способ запуска с помощью Spawner в интерактивном режиме:

```
<define name="#run_interactive" value="#spawner --separator=// -hr=1 --out
    =nul -wl=30 -tl=%time_limit -ml=%memory_limit -y=1 --// -sr=report.txt
    --in=*1.stdout --out=*1.stdin %interactor_name --// %deadline -sr= "/>
```

– default_interactor_name - имя программы интерактора по-умолчанию:

```
<define name="#default_interactor_name" value="Interact.exe" />
```

– в каждую секцию среды разработки, поддерживающую интерактивные задачи должен быть добавлен атрибут run_interactive:

```
<!-- Cross-platform C/C++ compiler -->
<de
    code="101"
    compile='#spawner #gnu_cpp -O2 "%full_name" -o "%name.exe"'
    run='#run "%name.exe"'
    run_interactive='#run_interactive "%name.exe"'
    runfile='%name.exe'
    generate='#spawner%redir %limits "%name.exe" %args'
    check='#spawner %limits "%name.exe" %checker_args'/>
```

– имя программы-интерактора по умолчанию может быть переопределено для среды разработки с помощью атрибута interactor_name, например для Java:

```
<!-- Java -->
<de
    code="401"
    compile='#spawner %comspec% /C ..\javac.cmd "%full_name"'
    run='#run "#java\java.exe" -Xss64m "%name"'
    interactor_name='"#java\java.exe" -Xss64m "Interact"'
    run_interactive='#run_interactive "#java\java.exe" -Xss64m "%name"'
    runfile='%name.class'
    generate='#spawner%redir %limits -wl:100 "#java\java.exe" -Xmx256m -
        Xss64m "%name" %args'
```

```
10      check='#spawner %limits "#java\java.exe" -Xss64m "%name" %checker_args
            '/>
```

5. Функциональные требования

Для интерактивных задач требуется поддерживать тот же функционал CATS что и для обычных задач[4]. Так как интерактивные задачи отличаются только методом запуска, то реализация этого функционала не потребует больших трудозатрат.

От компонента Spawner требуется:

- обеспечить возможность обменна данными между контроллером и нормалом посредством стандартных потоков ввода-вывода
- предоставить контроллеру
  - информацию о количестве нормалов в симуляции
  - возможность завершения указанного нормала
  - возможность определить индекс нормала, отправившего сообщение
  - возможность отсылать сообщение указанному нормалу
- корректно обрабатывать ситуацию прекращения работы нормала и продолжения симуляции

6. Проект

Подробное описание устройства проекта может быть найдено в [9]. В данном разделе будут рассмотрены внесённые изменения в различные модули, а так же описано и обосновано принятие некоторых решений.

6.1. Средства реализации

Для внесения изменений в компоненты cats-main и cats-judge использовался ЯП Perl, как основной язык системы. Изменения в Spawner вносились с

использованием ЯП C++11. Выбор данного стандарта C++ обоснован тем фактом, что актуальные версии популярных компиляторов C++ почти в полной мере поддерживают данный стандарт, а его использование позволяет более выразительно писать код, что уменьшает время разработки, а так же улучшает читаемость кода.

Для обеспечения возможности вывода stacktrace использовалась сторонняя библиотека stackwalker[21].

6.2. Интерактивные Задачи

Для поддержки интерактивных задач добавлено понятие метода запуска. Данное поле было добавлено в БД в таблицу Problems, в класс Problem компонента cats-main. Парсер формата задачи в компоненты cats-main был доработан для поддержки формата задачи с учётом разбора метода запуска. В cats-judge потребовалось добавить выбор метода запуска из XML файла настройки cats-judge с учётом метода запуска, указанного в задаче.

6.3. Поддержка многоагентных задач в Spawner

6.3.1. Рефакторинг Spawner

- разделение модуля main.cpp на:
  - main.cpp
  - spawner_base.hcpp
  - spawner_old.hcpp
  - spawner_new.hcpp
  - spawner_pcms2.hcpp
- удаление из проекта не используемых модулей
- замена символов табуляции на пробелы (в проекте была произвольно смешанная индентация)
- замена более чем одного идущего подряд перевода строки на один

– добавление символов перевода строки в конце файлов

– замена техники include guards на #pragma once

– переименование классов согласно стилю кодирования

– удалена более не используемая сторонняя библиотека json-cpp

6.3.2. Доработка модуля ошибок

Для автоматизации отчёта о критических ошибках во время работы Spawner использовался модуль error.hcpp, который предоставлял функцию raise_error. Данная функция принимала на вход произвольный класс и требовала от него наличия метода safe_release, который судя всего по задумке автора должен был освобождать ресурсы, занимаемые классом, в коде которого произошла ошибка. Ошибка всегда предполагалась вызванной некоторой функцией OS Windows, таким образом информация об ошибке всегда извлекалась с помощью GetLastEerror функции WinAPI.

В ходе работы данный модуль был полностью переписан. Добавлены макросы PANIC и PANIC_IF, которые в отличии от raise_error после вывода информации об ошибке немедленно прерывают работу программы. Реализация safe_release более не требуется, так как в случае немедленного завершения при критической ошибке нет смысла освобождать ресурсы, потому что критическая ошибка предполагает невозможность дальшнейшего выполнения. Так же в модуль error.hcpp добавлена возможность вывода stacktrace приложения в данный момент для нужд отладки.

Модуль предоставляет возможность установить с помощью функции set_on_panic_action действие, которое будет вызвано при вызове PANIC или PANIC_IF, в компоненте Spawner таким действием является вывод отчёта о выполнении.

При переписывании данного модуля была учтена возможность одновременного доступа к ресурсу хранящему сообщения об ошибках из нескольких потоков, а так же реализована защита от бесконечной рекурсии, когда критическая ошибка происходит во время вызова set_on_panic_action.

6.3.3. Завершение работы

В ходе изучения кода выяснилось, что не производится корректное завершение работы Spawner, так как для классов модуля pipe.hcpp и модуля buffer.hcpp никогда не осуществляется очистка ресурсов посредством вызова деструкторов классов при завершении программы.

Было решено использовать умные указатели предоставляемые стандартной библиотекой C++ для обёртки указателей на классы из вышеперечисленных модулей, что привело к вызову деструкторов данных классов при завершении программы.

В результате чего выяснилось, что программа зависает при завершении работы, так как код деструкторов классов input_pipe_c и output_pipe_c ожидает завершения работы потоков чтения и записи в системны пайпы. Данные потоки могли находиться в зависнувшем состоянии, так как в них использованы операции синхронного чтения и записи в системные объекты типа пайп. Данные операции не продолжат свою работу, пока над другим концом пайпа не будет осуществленна обратная операция. То есть если во время завершения выполнения некоторый поток осуществлял вход в синхронную функцию записи в некоторый пайп, а в это время программа исполняемая Spawner завершилась, то иногда могла возникунуть ситуация, что из этого пайпа уже некому читать информацию.

Для решения данной проблемы была использована функция WinAPI CancelSynchronousIo, которая отменяет операции синхронного ввода-вывода в пределах потока, что позволяет потоку продолжить исполнение и успешно завершиться.

6.3.4. Модуль pipe

Данный модуль реализует классы пайп и содержит реализацию потоков исполнения, осуществляющих чтение и запись из стандартных потоков ввода-вывода.

Был изменён алгоритм чтения пайпа стандартного вывода — данные буферизируютс до встречи первого символа разделителя строки (так как он является разделителем сообщения, согласно протоколу).

В класс выходного пайпа добавлен функтор process_message, который поз-

воляет перенести код обработки сообщений отсылаемых исполняемыми программами на более высокий уровень абстракции, в данном случае в модуль spawner_new.hcpp. Это было необходимо, так как для постобработки сообщений и мультиплексированиядемультиплексирования нужен контекст основного класса spawner_new_c. Данный код обработки сообщений активируется только в режиме управления и реализует протокол взаимодействия.

Был реализован алгоритм удаления буферов из действующих пайп с возможностью дальнейшего продолжения симуляции. Это необходимо при завершении какого-то процесса типа нормал, чтобы оставшиеся от него буфера не блокировали операции вводавывода.

Так как класс буфера двустороннего обмена данными был реализован с помощью анонимных пайп ОС Windows, операции записи и чтения виз него являются блокирующими. Существующий алгоритм опроса буферов потока исполнения записи STDIN пайпа предполагал последовательный и безошибочный доступ ко всем подключённым буферам. Для устранения этого ограничения была использована функция WinAPI PeekNamedPipe, которая позволяет асинхронно опросить пайп на предмет наличия данных.

6.3.5. Модуль mutex

Был реализован модуль mutex, предоставляющий удобный интерфейс к объекту ОС Windows Mutex, а так же содержащий полную проверку всех функций работы с мутексами на возвращаемые ошибки и снабжённый счётчиком ссылок. Стандартная библиотека C++11 содержит класс мутекс, предоставляющий похожий функционал, но данный класс не позволяет определить состояние Мутекса, опредляемое докмуентацией к WinAPI как WAIT_ABANDONED — потоки исполнения владевший мутексом не освободил Мутекс.

Данный модуль использался для защиты от одновременного доступа к ресурсам в ходе реализации протокола:

- Suspend/Resume thread
- buffer write access для экземпляра объекта buffer
- stdout buffer write - один мутекс на все экземпляры класса, так как они разделяют один ресурс (STDOUT Spawner)

– защита доступа к массиву буферов классов input_pipe_c и output_pipe_c.

– защита от выполнения более чем одного обработчика завершения процесса

7. Реализация и тестирование

Объём кода:

– cats-main, cats-judge: ЯП Perl — 200 строк кода, 10 коммитов

– Spawner: C++ — рефакторинг — 3172 строк кода, 5 коммитов

– Spawner: C++ — реализация протокола, исправление ошибок — 4202 строк кода, 43 коммита

Для тестирования поддержки интерактивных задач была разработана одна тривиальная интерактивная задача на сложение чисел, а так же успешно импортирована задача Gomoku [24]. В ходе реализации поддержки протокола в Spawner было использовано ручное тестирование, представляющее собой реализацию программ-участников протокола и многократные запуски этих программ с помощью Spawner для выявления ошибок типа race condition и deadlock. Исследовались различные сценарии некорректного поведения программ-участников протокола и удовлетворение требованиям безопасности Spawner в данных условиях.

Исходный код проекта открыт и доступен по адресу https://github.com/klenin/Spawner

Заключение

В результате данной работы в CATS появилась возможность использовать новый тип задач — интерактивные задачи, что выводит CATS на уровень аналогичных популярных решений. Был разработан протокол обмена данными для интерактивных многопользовательских задач и реализованна его поддержка в модуле контролируемого исполнения Spawner, что закладывает фундамент для дальнейшего развития системы. В модуле Spawner был произведён рефакторинг, а так же исправлены и задокументированны ошибки.

Дальнейшая работа в этом направлении заключается в проектировании и реализации поддержки многопользовательских интерактивных задач в другие компоненты CATS для проведения турниров ИИ. Что включает в себя разработку способа выбора участников сессии турнира, а так же системы оценки участников.

Планируется использование результатов данной работы в ходе летней практики 2015 года в рамках учебного процесса.

В ходе работы автором были углубленны знания в области системного программирования ОС Windows и получены навыки разработки многопоточных приложений, а так же осуществлено знакомство с ЯП Perl.

Список литературы

[1] Международная студенческая олимпиада по программированию ACM/ICPC — Wikipedia

[2] Steven S Skiena, Miguel A. Revilla, Programming Challenges: The Programming Contest Training Manual, 2002

[3] Система организации соревнований по программированию http://imcs.dvfu.ru/cats

[4] Рожков М., Кленин А.С. Дипломная работа «Система автоматического тестирования программ и организации соревнований по программированию». — ДВГУ, 2004г.

[5] Матвиенко В., Кленин А.С.Курсовая работа «Рендеринг математических выражений в MathML и HTML». — ДВГУ, 2005г.

[6] Коновалова Д., Кленин А.С. Курсовая работа «Поиск сходных алгоритмических конструкций в программном коде». — ДВГУ, 2005г.

[7] Перепечин В.В., Кленин А.С. Курсовая работа «AJAX-интерфейс для системы CATS». — ДВГУ, 2007г.

[8] Туфанов И.Е., Кленин А.С. Курсовая работа «Универсальный генератор тестов для системы CATS». — ДВГУ, 2008г.

[9] Храпченков П.Ф., Кленин А.С. Дипломная работа «Модуль контролируемого исполнения программ Spawner». — ДВФУ, 2014г.

[10] Модуль контролируемого исполнения программ Spawner, предыдущая версия http://imcs.dvgu.ru/cats/docs/spawner.html

[11] Всероссийская олимпиада школьников — Методические рекомендации. http://rosolymp.ru/index.php?option=com_content&view=article&id=6451&Itemid=909

[12] Система проведения соревнований EJudge http://ejudge.ru/

[13] Модуль контролируемого исполнения программ PCMS2 http://neerc.ifmo.ru/trains/information/software.html

[14] Система для проведения турниров и индивидуального решения задач по олимпиадному программированию http://www.contester.ru/

[15] PC^2(Programming Contest Control System) http://www.ecs.csus.edu/pc2/

[16] Олимпиады по программированию http://olympiads.ru/school/system/index.shtml

[17] Автоматизированная тестирующая система, для проведения соревнований по программированию http://domjudge.sourceforge.net/

[18] Универсальная система для проведения олимпиад по программированию http://code.google.com/p/dudge/

[19] Модуль контролируемого исполнения программ Spawner https://github.com/ShigiDono/Spawner

[20] Free Software Foundation, Inc. GNU General Public License. http://www.gnu.org/licenses/gpl.html. - 2007г.

[21] StackWalker - Walking the callstack. https://stackwalker.codeplex.com/

[22] Корнеев Г.А., Елизаров Р.А. Автоматическое тестирование решений на соревнованиях по программированию. «Телекоммуникации и информатизация» образования». 2003, №1 .– с. 61–73.

[23] E. Skochinski, J. Roberts, and M. Furon Posing Interactive Contest Problems in the ICPC Scoring Model ACM ICPC Competitive Learning Symposium April, 2008

[24] ACM ICPC 2014–2015, Northeastern European Regional Contest, Problems https://neerc.ifmo.ru/regional/neerc-2014-statements.pdf

[25] Robocode http://robocode.sourceforge.net/

[26] The AI Games http://theaigames.com/

[27] Vindinium http://vindinium.org/

[28] Rock Paper Scissors Contest http://www.rpscontest.com/

[29] Coding Game https://www.codingame.com/

[30] Code Cup http://www.codecup.nl/faq.php

[31] Hacker Rank https://www.hackerrank.com

[32] Docker, Network Configuration https://docs.docker.com/articles/networking/

[33] CATS, Формат заданий http://imcs.dvfu.ru/cats/docs/format.html